

Simulation of Turing machines with analytic discrete ODEs: **FPTIME** and **FPSPACE** over the reals characterised with discrete ordinary differential equations

MANON BLANC
OLIVIER BOURNEZ

Abstract: We prove that functions over the reals computable in polynomial time can be characterised using discrete ordinary differential equations (ODE), also known as finite differences. We also provide a characterisation of functions computable in polynomial space over the reals.

In particular, this covers space complexity, while existing characterisations were only able to cover time complexity, and were restricted to functions over the integers. We prove furthermore that no artificial sign or test function is needed even for time complexity.

At a technical level, this is obtained by proving that Turing machines can be simulated with analytic discrete ordinary differential equations. We believe this result opens the way to many applications, as it opens the possibility of programming with ODEs, with an underlying well-understood time and space complexity.

2000 Mathematics Subject Classification 03D15, 03D20, 68Q15, 68Q19, 39A05, 26E05
(primary)

Keywords: Discrete ordinary differential equations, Finite Differences, Implicit complexity, Recursion scheme, Ordinary differential equations, Models of computation, Analog Computations

1 Introduction

Recursion schemes constitute a major approach to classical computability theory and, to some extent, to complexity theory. The foundational characterisation of **FPTIME**, based on bounded primitive recursion on notations, due to Cobham [16] gave birth to the field of *implicit complexity* at the interplay of logic and theory of programming. Alternative characterisations, based on safe recursion [1] or on ramification ([23, 22]) or for other classes [24] followed: see [14, 15] for monographs.

Initially motivated to help understand how analogue models of computations compare to classical digital ones, in an orthogonal way, various computability and complexity classes have been recently characterised using Ordinary Differential Equations (ODE). An unexpected side effect of these proofs is the possibility of programming with classical ODEs, over the continuum. It recently led to solving various open problems. This includes the proof of the existence of a universal ODE [10], the proof of the Turing-completeness of chemical reactions [17], or hardness of problems related to dynamical systems [19]. While the above results are easy to state, their proofs are mixing considerations about approximations, control of errors, and various constructions to emulate continuously some discrete processes, despite some recent attempts for a kind of programming languages to help intuition [6].

Discrete ODEs, that we consider in this article, are an approach in-between born from the attempt of [8, 9] to explain some of the constructions for continuous ODEs in an easier way. The basic principle is, for a function $\mathbf{f}(x)$, to consider its discrete derivative defined as $\Delta\mathbf{f}(x) = \mathbf{f}(x + 1) - \mathbf{f}(x)$ (also denoted $\mathbf{f}'(x)$ in what follows to help analogy with classical continuous counterparts). A consequence of this attempt is the characterisation obtained in [8, 9]. They provided a characterisation of **FPTIME** for functions over the integers that does not require the specification of an explicit bound in the recursion, in contrast to Cobham's work [16], nor the assignment of a specific role or type to variables, in contrast to safe recursion or ramification [1, 21]. Instead, they only assume involved ODEs to be linear, a very classical natural concept for differential equations. A characterisation, like ours, happens to be rather simple using only common notions from the world of ODEs. In particular, considering *linear* ordinary differential equations is very natural for ODEs.

Remark 1 Unfortunately, even if it was the original motivation, both approaches for characterising complexity classes for continuous and discrete ODEs are currently not directly connected. A key difference is that there is no simple expression (no analogue of the Leibniz rule) for the derivative of the composition of functions in discrete settings. The Leibniz rule is a very basic tool for establishing results over the continuum, using various stability properties, but similar statements cannot be established easily over discrete settings.

In the context of algebraic classes of functions, the following notation is classical: call *operation* an operator that takes finitely many functions and returns some new function defined from them. Then $[f_1, f_2, \dots, f_k; op_1, op_2, \dots, op_\ell]$ denotes the smallest set of functions containing f_1, f_2, \dots, f_k that is closed under the operations $op_1, op_2, \dots, op_\ell$. Call *discrete function* a function of type $f : S_1 \times \dots \times S_d \rightarrow S'_1 \times \dots S'_{d'}$, where

each S_i, S'_i is either \mathbb{N} or \mathbb{Z} . Write **FPTIME** for the class of functions computable in polynomial time, and **FPSPACE** for the class of functions computable in polynomial space.

The literature considers two possible definitions for **FPSPACE**, according to whether functions with non-polynomial size values are allowed or not. In our case, we should add “whose outputs remain of polynomial size”, to resolve the ambiguity.

Remark 2 The point is that, otherwise, the class is not closed by composition. This may be considered as a basic requirement when talking about the complexity of functions. The issue is about the usage of not counting the output as part of the total space used. In this model, given f computable in polynomial space, and g in logarithmic space, $f \circ g$ (and $g \circ f$) is computable in polynomial space. But this is not true if we assume only f and g to be computable in polynomial space, since the first might give an output of exponential size.

A main result of [8, 9] is the following (\mathbb{LDL} stands for linear derivation on length):

Theorem 1.1 ([8]) *For functions over the reals, we have $\mathbb{LDL} = \mathbf{FPTIME}$ where*

$$\mathbb{LDL} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \text{sg}(x); \text{composition, linear length ODE}].$$

In particular, writing as usual B^A for functions from A to B , we deduce:

Corollary 1.2 (Functions over the integers) $\mathbb{LDL} \cap \mathbb{N}^\mathbb{N} = \mathbf{FPTIME} \cap \mathbb{N}^\mathbb{N}$.

That is to say, \mathbb{LDL} (and hence **FPTIME** for functions over the integers) is the smallest class of functions that contains the constant functions $\mathbf{0}$ and $\mathbf{1}$, the projections π_i^k of the i^e coordinate of a vector of size k , the length function $\ell(x)$, mapping an integer to the length of its binary representation, the addition $x+y$, the subtraction $x-y$, the multiplication $x \times y$, the **sign** function $\text{sg}(x)$ and that is closed under composition (when defined) and linear length-ODE scheme: the linear length-ODE scheme, formally given by Definition 2.4, corresponds to defining a function from a linear ODE with respect to derivation along the length of the argument, so of the form

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{A}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}].$$

Here, we use the notation $\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell}$ which corresponds to the derivation of \mathbf{f} along the length function: given some function $\mathcal{L} : \mathbb{N}^{p+1} \rightarrow \mathbb{Z}$ and in particular for the case where $\mathcal{L}(x, \mathbf{y}) = \ell(x)$,

$$(1-1) \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}(x, \mathbf{y})} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y})$$

is a formal synonym for

$$\mathbf{f}(x+1, \mathbf{y}) = \mathbf{f}(x, \mathbf{y}) + (\mathcal{L}(x+1, \mathbf{y}) - \mathcal{L}(x, \mathbf{y})) \cdot \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}).$$

Remark 3 This concept introduced in [8, 9], is motivated by the fact that the latter expression is similar to the classical formula for continuous ODEs:

$$\frac{\delta f(x, \mathbf{y})}{\delta x} = \frac{\delta \mathcal{L}(x, \mathbf{y})}{\delta x} \cdot \frac{\delta f(x, \mathbf{y})}{\delta \mathcal{L}(x, \mathbf{y})},$$

and hence is similar to a change of variable. Consequently, a linear length-ODE is a linear ODE over a variable t once the change of variable $t = \ell(x)$ is done.

However, in the context of (classical) ODEs, considering functions over the reals is more natural than only functions over the integers. Call *real function* a function $f : S_1 \times \dots \times S_d \rightarrow S'_1 \times \dots \times S'_{d'}$, where each S_i, S'_i is either \mathbb{R} , \mathbb{N} or \mathbb{Z} . A natural question about the characterisation of **FPTIME** for real functions arises, and not only discrete functions: we consider here computability over the reals in its most classical approach, namely computable analysis [30].

As a first step, the class

$$\mathbb{LDL}^\bullet = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\text{cond}}(x), \frac{x}{2}; \text{composition, linear length ODE}]$$

has been considered in [3, 2] where the authors get some characterisation of **PTIME**, but only for functions from the integers to the reals (i.e. sequences) while it would be more natural to characterise functions from the reals to the reals.

More importantly, this was obtained by assuming that some **non-analytic exact function** is among the basic available functions to simulate a Turing machine: $\overline{\text{cond}}$ valuing 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$.

We prove first this is not needed, and mainly, we extend all previous results to real functions, furthermore covering not only time complexity but also space complexity. Consider

$$\mathbb{LDL}^\circ = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE}],$$

where $\ell : \mathbb{N} \rightarrow \mathbb{N}$ is the length function, mapping some integer to the length of its binary representation, $\frac{x}{2} : \mathbb{R} \rightarrow \mathbb{R}$ is the function dividing by 2 (similarly for $\frac{x}{3}$) and all other basic functions defined exactly as for \mathbb{LDL} , but are considered here as functions from the reals to reals.

Remark 4 This class is LDL but without the $\text{sg}(x)$ function, nor the multiplication function, or $\overline{\text{LDL}}^\bullet$ but without the cond function, nor the multiplication. This is done by adding the analytic \tanh functions as a substitute (and adding $x/3$).

Remark 5 We can consider $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R}$ but as functions may have different types of outputs, the composition is an issue. We consider, as this is done in [3, 2], that composition may not be defined in some cases: it is a partial operator. For example, given $f : \mathbb{N} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$, the composition of g and f is defined as expected, but f cannot be composed with a function such as $h : \mathbb{N} \rightarrow \mathbb{N}$.

First, we improve Theorem 1.1 by stating **FPTIME** over the integers can be characterised algebraically using linear length ODEs and only analytic functions (i.e. no need for sign function). Since LDL° is about functions over the reals, and Theorem 1.1 is about functions over the integers, we need a way to compare these classes. Given a function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of \mathbb{N}^d , say at a distance less than $1/4$, we write $\text{DP}(\mathbf{f})$ for its discrete part: this is the function from $\mathbb{N}^d \rightarrow \mathbb{N}^{d'}$ mapping $\mathbf{n} \in \mathbb{N}^d$ to the integer rounding of $\mathbf{f}(n)$. Given a class \mathcal{C} of such functions, we write $\text{DP}(\mathcal{C})$ for the class of the discrete parts of the functions of \mathcal{C} .

Theorem 1.3 $\text{DP}(\text{LDL}^\circ) = \text{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$.

Second, we improve [3]: Write $\overline{\text{LDL}}^\circ$ for the class obtained by adding some effective limit operation similar to the one considered in [3] to get $\overline{\text{LDL}}^\bullet$. We get a characterisation of functions over the reals (and not only sequences as in [3]) computable in polynomial time.

Theorem 1.4 (Generic functions over the reals) $\overline{\text{LDL}}^\circ \cap \mathbb{R}^{\mathbb{R}} = \text{FPTIME} \cap \mathbb{R}^{\mathbb{R}}$
 More generally: $\overline{\text{LDL}}^\circ \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \text{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}$.

We also prove that, by adding a robust linear ODE scheme (Definition 2.7), we get a class RLD° (this stands for robust linear derivation) with similar statements but for **FPSPACE**.

Theorem 1.5 $\text{DP}(\text{RLD}^\circ) = \text{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

Theorem 1.6 (Generic functions over the reals) $\overline{\text{RLD}}^\circ \cap \mathbb{R}^{\mathbb{R}} = \text{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$
 More generally: $\overline{\text{RLD}}^\circ \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \text{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}$.

As far as we know, this is the first time a characterisation of **FPSpace** with discrete ODEs is provided. If we forget the context of discrete ODEs, **FPSpace** has been characterised in [29] but using a bounded recursion scheme, i.e. requiring some explicit bound in the spirit of Cobham’s statement [16]. We avoid this issue by considering numerically stable schemes, which are very natural in the context of ODEs.

At a technical level, all our results are obtained by proving Turing machines can be suitably simulated with analytic discrete ODEs. We believe our constructions could be applied to many other situations, where programming with ODEs is needed.

More on related works In addition to the previous state-of-the-art discussions, we comment here on some aspects: Our ways of simulating Turing machines have some reminiscence of similar constructions used in other contexts such as Neural Networks [28, 27]. In particular, we use a Cantor-like encoding set \mathcal{I} with inspiration from these references. These references use some particular sigmoid function σ (called sometimes the *saturated linear function*) that values 0 when $x \leq 0$, x for $0 \leq x \leq 1$, 1 for $x \geq 1$. The latter is equivalent to $\overline{\text{cond}}(\frac{1}{4} + \frac{1}{2}x)$, for the function considered in [3, 2] and hence their constructions can be reformulated using the $\overline{\text{cond}}$ function. We completely avoid this, by considering the \tanh function, which is more natural in the context of formal neural networks. The models considered in [28, 27] are recurrent, while our constructions are not recurrent neural networks, and second, their models are restricted to live on the compact domain $[0, 1]$, which forbids getting functions from $\mathbb{R} \rightarrow \mathbb{R}$, while our settings allow more general functions. Our proofs also require functions taking some integer arguments, that would be impossible to consider in their settings (unless at the price of an artificial recoding).

Remark 6 In some sense, our constructions can be seen as operators that map to a family of neural networks in the spirit of these models, instead of considering fixed recurrent neural networks, but also dealing with \tanh , and not requiring the saturated linear function.

The question of whether Turing machines can be simulated by recurrent neural networks of finite size, using the \tanh activation function (and not the “exact” saturated linear function) is a well-known long-standing open problem. Despite some attempts, such as the one described in [27], up to our knowledge, there remains some of the statements not yet formally proved, or at least not fully generally accepted, in the existing proofs. Our statements are dealing with the \tanh activation function, in some sense, but we avoid this open question by restricting it to finite space or time computations. By the

way, our proofs state this is possible if the space or the time of the machine is bounded, up to some controlled error.

In the context of neural network models, there have been several characterisations of complexity classes over the discrete (see e.g. the monograph [27] about the approach discussed above, but not only), as far as we know, the relation between formal neural networks with classes of computable analysis has never been established before.

If we do not restrict ourselves to neural network-related models, as in all these previous contexts, as far as we know, only a few papers have been devoted to characterisations of complexity, and even computability, classes in the sense of computable analysis. There have been some attempts using continuous ODEs [7], that we already mentioned, or the so-called \mathbb{R} -recursive functions [11]. For discrete schemata, we only know [12] and [25], focusing on computability and not complexity.

Organisation of the article In Section 2, we recall some basic statements about the theory of discrete ODEs. More details can be found in Appendix B, mostly repeated from [8, 9, 3], to be self-content. In Section 3, we establish some properties about particular functions required for our proofs. In Section 4 we prove our main technical result: Turing machines can be simulated using functions from LDL° . Section 5 is about converting integers and reals (dyadic) to words of a specific form. Section 6 is about applications of our toolbox. We prove in particular all the above theorems.

About appendices In this article, when we say that a function $f : S_1 \times \dots \times S_d \rightarrow \mathbb{R}^{d'}$ is (respectively: polynomial time or space) computable this will always be in the sense of computable analysis: see e.g. [13, 30]. As we did not find a reference where the case of functions mixing integer and real arguments is fully formalised, we proposed a formalisation in [3]. In order to be self-content, we repeat it in Appendix A. In order not to expect our readers to be familiar with the theory of discrete ODEs, as we already wrote, we repeat some basic statements in Appendix B, mostly repeated from [8, 9]. As we need some results from [3], we also repeat their proof in Appendix C.

Note about current version vs final version This article is the journal version of an article accepted at MFCS’2023 [4].

2 Some concepts related to discrete ODEs

In this section, we recall some concepts and definitions from discrete ODEs, either well-known or established in [8, 9, 3]: more details, repeated from these references, are provided in Appendix B.

In order to get a uniform presentation, we consider here that \tanh is \tanh , the hyperbolic tangent. The papers [8, 9] use similar definitions with the sign function sg and [3] with the piecewise affine function $\overline{\text{cond}}$, which values 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$, instead of \tanh .

Definition 2.1 ([3]) A \tanh -polynomial expression $P(x_1, \dots, x_h)$ is an expression built-on $+$, $-$, \times (often denoted \cdot) and \tanh functions over a set of variables $V = \{x_1, \dots, x_h\}$ and integer constants.

We need to measure the degree, similarly to the classical notion of degree in a polynomial expression, but considering all subterms that are within the scope of a \tanh function contributes to 0 to the degree.

Definition 2.2 ([3]) The degree $\deg(x, P)$ of a term P in $x \in V$ is defined inductively as follows:

- $\deg(x, x) = 1$ and for $x' \in V \cup \mathbb{Z}$ such that $x' \neq x$, $\deg(x, x') = 0$;
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$;
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$;
- $\deg(x, \tanh(P)) = 0$.

A \tanh -polynomial expression P is *essentially constant* in x if $\deg(x, P) = 0$.

A vectorial function (resp. a matrix or a vector) is said to be a \tanh -polynomial expression if all its coordinates (resp. coefficients) are, and *essentially constant* if all its coefficients are.

Definition 2.3 ([8, 9, 3]) A \tanh -polynomial expression $\mathbf{g}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ is *essentially linear* in $\mathbf{f}(x, \mathbf{y})$ if it is of the form: $\mathbf{A}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}]$ where \mathbf{A} and \mathbf{B} are \tanh -polynomial expressions essentially constant in $\mathbf{f}(x, \mathbf{y})$.

For example,

- the expression $P(x, y, z) = x \cdot \tanh(x^2 - z) \cdot y + y^3$ is essentially linear in x , essentially constant in z and not linear in y .
- The expression $P(x, 2^{\ell(y)}, z) = \overline{\text{cond}}(x^2 - z) \cdot z^2 + 2^{\ell(y)}$ is essentially constant in x , essentially linear in $2^{\ell(y)}$ (but not essentially constant) and not essentially linear in z .
- The expression: $z + (1 - \tanh(x)) \cdot (1 - \tanh(-x)) \cdot (y - z)$ is essentially constant in x and linear in y and z .

Definition 2.4 (Linear length ODE [8, 9]) A function \mathbf{f} is linear length-ODE definable from \mathbf{u} *essentially linear* in $\mathbf{f}(x, \mathbf{y})$, \mathbf{g} and \mathbf{h} if it corresponds to the solution of

$$(2-1) \quad f(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}).$$

A fundamental fact is that the derivation with respect to length provides a way to do some change of variables:

Lemma 2.5 ([8, 9]) Assume that (2–1) holds. Then $\mathbf{f}(x, \mathbf{y})$ is given by $\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\ell(x), \mathbf{y})$ where \mathbf{F} is the solution of the initial value problem

$$(2-2) \quad \mathbf{F}(1, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \frac{\partial \mathbf{F}(t, \mathbf{y})}{\partial t} = \mathbf{u}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y}).$$

This means $\mathbf{f}(x, \mathbf{y})$ depends only on the length of its first argument: $\mathbf{f}(x, \mathbf{y}) = \mathbf{f}(2^{\ell(x)}, \mathbf{y})$. Then (2–2) can be seen as defining a function (with this latter property) by a recurrence of type

$$(2-3) \quad \mathbf{f}(2^0, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \mathbf{f}(2^{t+1}, \mathbf{y}) = \bar{\mathbf{u}}(\mathbf{f}(2^t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t, \mathbf{y}).$$

for some $\bar{\mathbf{u}}$ is *essentially linear* in $\mathbf{f}(2^t, \mathbf{y})$. As recurrence (2–2) is basically equivalent to (2–1):

Corollary 2.6 (Linear length ODE presented with powers of 2) A function \mathbf{f} is linear \mathcal{L} -ODE definable iff the value of $\mathbf{f}(x, \mathbf{y})$ depends only on the length of its first argument and satisfies (2–3), for some \mathbf{g} and \mathbf{h} , and $\bar{\mathbf{u}}$, *essentially linear* in $\mathbf{f}(2^t, \mathbf{y})$.

We guess it is easier for our reader to deal with recurrences of the form (2–3) than with ODEs of the form (2–1). Consequently, this is how we will describe many functions from now on, starting with some basic functions, authorising compositions, and the above schemes. As an example, $n \mapsto 2^n$ can easily be defined that way. Consider:

$$\begin{cases} 2^0 &= 1 \\ 2^{n+1} &= 2 \cdot 2^n = 2^n + 2^n \end{cases}$$

Similarly, we can produce $n \mapsto 2^{p(n)}$ for any polynomial p . For example,

$$(n_1, \dots, n_k) \rightarrow 2^{n_1 n_2 \dots n_k}$$

can be obtained, using k such schemes in turn, providing the case of the polynomial $p(n) = n^k$.

When talking about space complexity, we will also consider the case where the ODE is not derivated with respect to length but with classical derivation. For functions over the reals, an important issue is numerical stability.

Definition 2.7 (Robust linear ODE [8, 9]) A bounded function \mathbf{f} is robustly linear ODE definable from \mathbf{u} *essentially linear* in $\mathbf{f}(x, \mathbf{y})$, \mathbf{g} and \mathbf{h} if:

- (1) it corresponds to the solution of

$$(2-4) \quad \mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}),$$

- (2) where the schema (2-4) is polynomially numerically stable.

Here, writing $a =_n b$ for $\|a - b\| \leq 2^{-n}$ for conciseness, 2. means formally there exists some polynomial p such that, for all integer n , writing $\epsilon(n) = p(n + \ell(\mathbf{y}))$, if you consider any solution of

$$\begin{cases} \tilde{\mathbf{y}} &=_{\epsilon(n)} \mathbf{y} \\ \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}) &=_{\epsilon(n)} \mathbf{h}(x, \tilde{\mathbf{y}}) \\ \tilde{\mathbf{f}}(0, \tilde{\mathbf{y}}) &=_{\epsilon(n)} \mathbf{g}(\mathbf{y}) \\ \frac{\partial \tilde{\mathbf{f}}(x, \tilde{\mathbf{y}})}{\partial x} &=_{\epsilon(n)} \mathbf{u}(\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}), x, \tilde{\mathbf{y}}) \end{cases}$$

then

$$\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{f}(x, \mathbf{y}).$$

This corresponds roughly to the concept of polynomially robust to precision considered in [5], and turns out to be a very natural concept for functions defined on a compact.

Remark 7 For linear length ODEs, we did not have to put explicitly numerical stability as a hypothesis, as it comes free from the fact that we consider solutions at most at some logarithmic value of their arguments. But this is required here to guarantee the computability of the solution (and even polynomial space computability).

Remark 8 Notice that, over the continuum, even computable ODEs may have no computable solution [26]. Over the discrete, not all dynamics can be simulated, and numerical stability is indeed an issue.

Remark 9 We believe the hypothesis that \mathbf{f} is bounded can be relaxed to: “the function does not grow as much as the exponential of a polynomial in the length of its arguments”, i.e. not more than a function of **FPSPACE**, from arguments similar to the ones of [29] about functions over the integers.

3 Some results about various functions

A key part of our proofs is the construction of very specific functions in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$: we write $\{x\}$ for the fractional part of the real x , i.e. $\{x\} = x - \lfloor x \rfloor$.

A first observation is that we can uniformly approximate the $\text{ReLU}(x) = \max(0, x)$ function using an essentially constant function:

Lemma 3.1 Consider (see Figure 1)

$$Y(x, 2^{m+2}) = \frac{1 + \tanh(2^{m+2}x)}{2}$$

For all integer m , for all $x \in \mathbb{R}$,

$$|\text{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}.$$

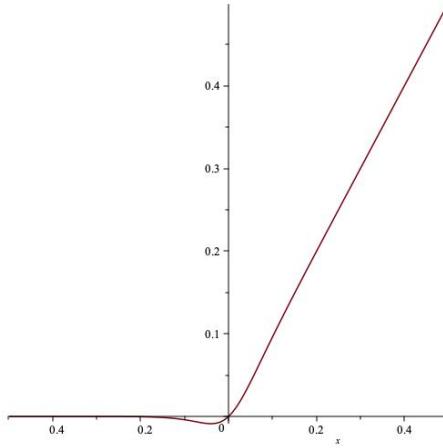


Figure 1: Graphical representation of $xY(x, 2^{2+2})$ obtained with maple.

To prove Lemma 3.1, we start by the following basic facts about function \tanh .

Lemma 3.2 $|1 + \tanh x| \leq 2 \exp(-2|x|)$ for $x \in (-\infty, 0]$.

Proof For $x \leq 0$, $|1 + \tanh x| = 1 + \tanh x$, and $|x| = -x$. We have $f(x) = 2 \exp(2x) - \tanh(x) - 1 = 2 \exp(2x) - \frac{1 - \exp(-2x)}{1 + \exp(-2x)} - 1 = \frac{2 \exp(4x)}{1 + \exp(2x)} \geq 0$. \square

Lemma 3.3 $|1 - \tanh x| \leq 2 \exp(-2|x|)$ for $x \in [0, +\infty)$.

Proof This follows from Lemma 3.2, using the fact that \tanh is odd. \square

Proof of Lemma 3.1 Let $m \in \mathbb{N}$. Consider $Y(x, K) = \frac{1 + \tanh(Kx)}{2}$, with $K > 0$.

For $0 \leq x$, $\text{ReLU}(x) = x$, and $|\text{ReLU}(x) - xY(x, K)| = \frac{x}{2}|1 - \tanh(Kx)| \leq x \exp(-2Kx)$ from Lemma 3.3.

For $x \leq 0$, $\text{ReLU}(x) = 0$, and $|\text{ReLU}(x) - xY(x, K)| = \frac{|x|}{2}|1 + \tanh(Kx)| \leq |x| \exp(-2K|x|)$ from Lemma 3.2, which is the same expression as above for $0 \leq x$.

Function $g(x) = x \exp(-2Kx)$ has its derivative $g'(x) = \exp(-2Kx)(1 - 2Kx)$. We deduce the maximum of this function $g(x)$ over \mathbb{R} is in $\frac{1}{2K}$, and that the maximum value of $g(x)$ is $\frac{1}{e2K}$.

Consequently, if we take $K = 2^{m+2}$, then $g(x) \leq 2^{-m}$ for all x , and we conclude. \square

We deduce we can uniformly approximate the continuous sigmoid functions (when $1/(b-a)$ is in \mathbb{LDL}°) defined as: $s(a, b, x) = 0$ whenever $w \leq a$, $\frac{x-a}{b-a}$ whenever $a \leq x \leq b$, and 1 whenever $b \leq x$.

Lemma 3.4 (Uniform approximation of any piecewise continuous sigmoid) Assume $a, b, \frac{1}{b-a}$ is in \mathbb{LDL}° . Then there is some function (illustrated by Figure 2) $\mathcal{C}\text{-}s(z, a, b, x) \in \mathbb{LDL}^\circ$ such that for all integer m ,

$$|\mathcal{C}\text{-}s(2^m, a, b, x) - s(a, b, x)| \leq 2^{-m}.$$

Proof We can write $s(a, b, x) = \frac{\text{ReLU}(x-a) - \text{ReLU}(x-b)}{b-a}$. Consider $\mathcal{C}\text{-}s(z, a, b, x) = \frac{(x-a)Y(x-a, z2^{1+c}) - (x-b)Y(x-b, z2^{1+c})}{b-a}$. Thus, $|\mathcal{C}\text{-}s(2^{m+1+c}, a, b, x) - s(a, b, x)| \leq \frac{2 \cdot 2^{-m-1-c}}{b-a}$, using the triangle inequality. Take c such that $\frac{1}{b-a} \leq 2^c$. \square

The existence of the following function will play an important role to obtain some various other functions.

Theorem 3.5 There exists some function (illustrated by Figure 3) $\xi : \mathbb{N}^2 \rightarrow \mathbb{R}$ in \mathbb{LDL}° such that for all $n, m \in \mathbb{N}$ and $x \in [-2^n, 2^n]$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$,

$$\left| \xi(2^m, 2^n, x) - \left\{ x - \frac{1}{8} \right\} \right| \leq 2^{-m}.$$

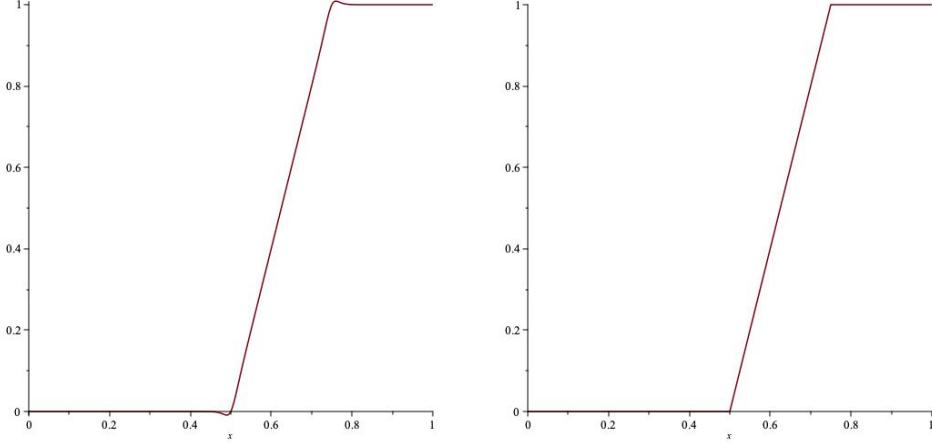


Figure 2: Graphical representation of $\mathcal{C}\text{-s}(2, \frac{1}{2}, \frac{3}{4}, x)$ and $\mathcal{C}\text{-s}(2^5, \frac{1}{2}, \frac{3}{4}, x)$ obtained with maple.

Proof If we take ξ' that satisfies the constraint only when $x \geq 0$, and that values 0 for $x \leq 0$, then $\frac{3}{4} - \xi'(\cdot, \cdot, -x)$ would satisfy the constraint when $x \leq 0$, but values $3/4$ for $x \geq 0$. So,

$$\xi(2^m, N, x) = \xi'(2^{m+2}, N, x) - \xi'(2^{m+2}, N, -x) + \frac{3}{4} - \frac{3}{4} \mathcal{C}\text{-s}(2^{m+2}, 0, \frac{1}{8}, x)$$

would work for all x . So it remains to construct ξ' such that for all $n \in \mathbb{N}$, $x \in [0, 2^n]$ and $m \in \mathbb{N}$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, $|\xi'(2^m, 2^n, x) - \{x - \frac{1}{8}\}| \leq 2^{-m}$, and $|\xi'(2^m, N, x) - 0| \leq 2^{-m}$ for $x \leq 0$.

Let $s(x) = \frac{3}{4} \text{s}(\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}, x)$. Over $[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, we have $s(x) = \{x - \frac{1}{8}\}$. Actually, we will even construct ξ' with the stronger properties that whenever $x \in [\lfloor x \rfloor + \frac{1}{8} - 2^{-m}, \lfloor x \rfloor + \frac{7}{8} + 2^{-m}]$, $|\xi'(2^m, 2^n, x) - s(x - \frac{1}{8})| \leq 2^{-m}$.

It suffices to define ξ' by induction by

$$\begin{cases} \xi'(2^m, 2^0, x) &= \frac{3}{4} \mathcal{C}\text{-s}(2^m, \frac{1}{8}, \frac{7}{8}, x) \\ \xi'(2^m, 2^{n+1}, x) &= \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x)) \end{cases}$$

where

$$F(2^{m+1}, K, x) = x - K \cdot \mathcal{C}\text{-s}(2^{m+1}, K + \frac{1}{32}, K + \frac{3}{32}, x).$$

Let $I_{\lfloor x \rfloor}$ be $[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, $x \in I_{\lfloor x \rfloor}$, and let us first study the value of $F(2^{m+1}, 2^n, x)$:

- If $x \leq 2^n$, by definition of $\mathcal{C}\text{-s}$, $|F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$, with $x \in I_{\lfloor x \rfloor}$.

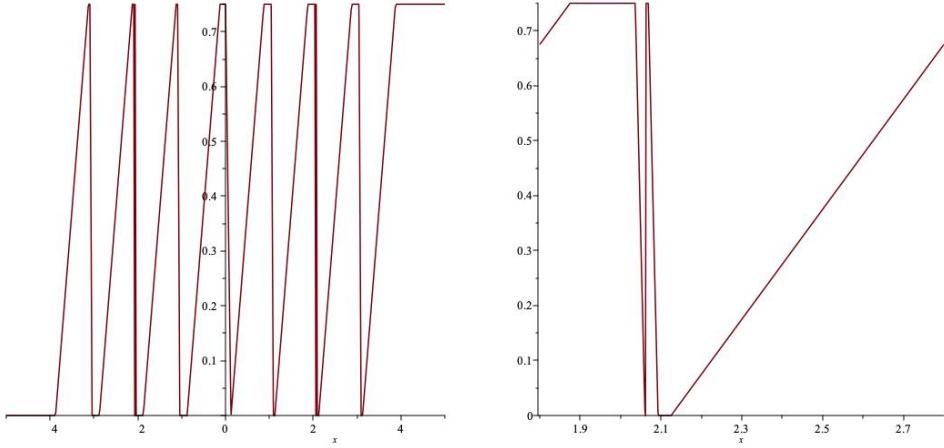


Figure 3: Graphical representations of $\xi(2, 4, x)$ obtained with maple: some details on the right.

- If $2^n + \frac{1}{8} \leq x$ then $|F(2^{m+1}, 2^n, x) - (x - 2^n)| \leq 2^{-(m+1)}$ with $x - 2^n \in I_{\lfloor x \rfloor - 2^n}$.

Now, the property is true by induction. Indeed, it is true for $n = 0$ by definition of $\xi'(2^m, 2^0, x)$. We now assume it is true for some $n \in \mathbb{N}$. We have $\xi'(2^m, 2^{n+1}, x) = \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x))$. Thus, by induction hypothesis, $|\xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x)) - s(F(2^{m+1}, 2^n, x) - 1/8)| \leq 2^{-(m+1)}$. Now:

- If $x \leq 2^n$, by definition of $\mathcal{C}\text{-}\mathfrak{s}$, $|F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$, and as s is 1-Lipschitz, $|s(F(2^{m+1}, 2^n, x) - \frac{1}{8}) - s(x - \frac{1}{8})| \leq |F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$. Consequently, $|\xi'(2^m, 2^{n+1}, x) - s(x - \frac{1}{8})| \leq 2^{-m}$ and the property holds for $n + 1$.
- If $2^n + \frac{1}{8} \leq x$ then $|F(2^{m+1}, 2^n, x) - (x - 2^n)| \leq 2^{-(m+1)}$ and as s is 1-Lipschitz, $|s(F(2^{m+1}, 2^n, x) - \frac{1}{8}) - s(x - 2^n - \frac{1}{8})| \leq |F(2^{m+1}, 2^n, x) - x - 2^n| \leq 2^{-(m+1)}$. Consequently, $|\xi'(2^m, 2^{n+1}, x) - s(x - 2^n - \frac{1}{8})| \leq 2^{-m}$ and the property holds for $n + 1$.

There remains to prove that the function ξ' is in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$. Unfortunately, this is not clear from the recursive definition, but this can be written in another way, from which this follows. Indeed, we have from an easy induction that $\xi'(2^m, 2^n, x) = F(2^{m+n-1}, 2^0, F(2^{m+n-2}, 2^1, F(2^{m+n-3}, 2^2 (\dots, F(2^m, 2^{n-1}, x))))$, if we define

$$F(2^m, 2^0, x) = \xi'(2^m, 2^0, x) = \frac{3}{4} \mathcal{C}\text{-}\mathfrak{s}(2^m, \frac{1}{8}, \frac{7}{8}, x).$$

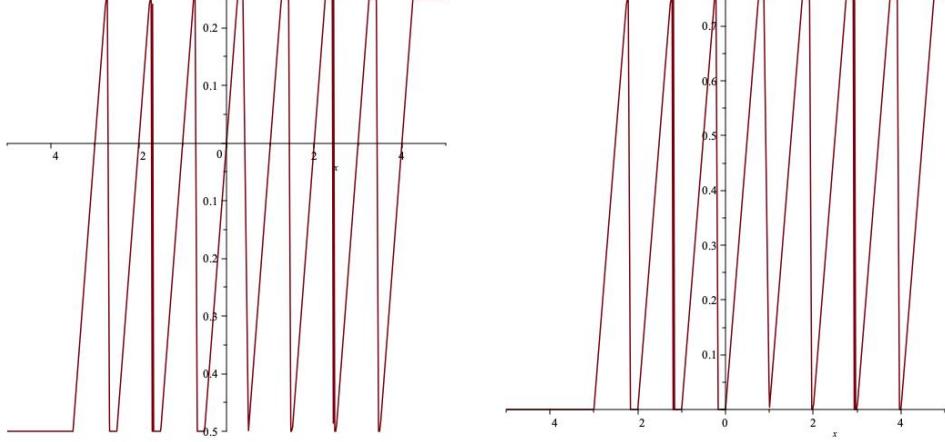


Figure 4: Graphical representation of $\xi_1(2, 4, x)$ and $\xi_2(2, 4, x)$ obtained with maple.

Then, we can obtain $\xi'(2^m, 2^n, x) = H(2^m, 2^{n-1}, 2^n, x)$ with

$$\begin{aligned} H(2^m, 2^0, 2^n, x) &= F(2^m, 2^{n-1}, x) \\ H(2^m, 2^{t+1}, 2^n, x) &= F(2^{m+t}, 2^{n-1-t}, H(2^m, 2^t, 2^n, x)) \\ &= H(2^m, 2^t, 2^n, x) - 2^{n-1-t} \cdot \mathcal{C}\text{-}\mathfrak{s}(2^{m+t}, 2^{n-1-t}, 2^{n-1-t}) \\ &\quad + \frac{1}{8}, H(2^m, 2^t, 2^n, x)) \end{aligned}$$

Such a recurrence can be then seen as a linear length ordinary differential equation, in the length of its first argument. It follows that ξ' is in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$. \square

From the construction of the previous functions, we obtain a bestiary of various functions

Corollary 3.6 *There exists (see Figure 4) $\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that, for all $n, m \in \mathbb{N}$, $[x] \in [-2^n + 1, 2^n]$,*

- whenever $x \in [x] - \frac{1}{2}, [x] + \frac{1}{4}]$,

$$|\xi_1(2^m, 2^n, x) - \{x\}| \leq 2^{-m},$$

- and whenever $x \in [x], [x] + \frac{3}{4}]$,

$$|\xi_2(2^m, 2^n, x) - \{x\}| \leq 2^{-m}.$$

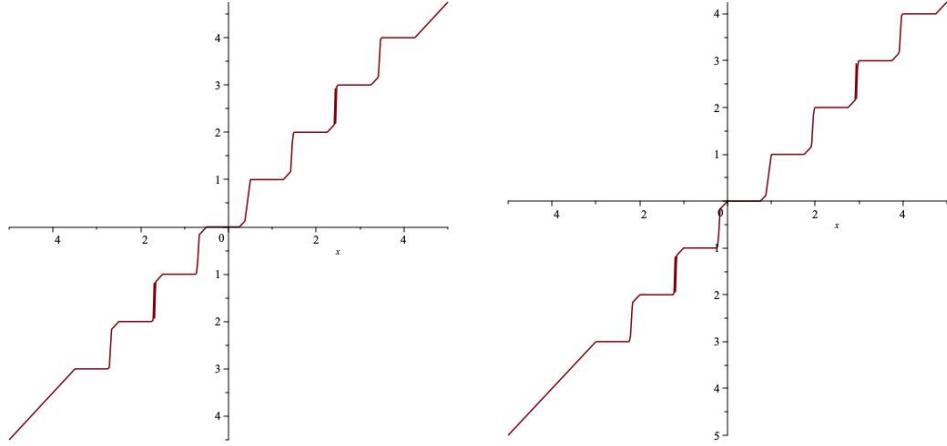


Figure 5: Graphical representation of $\sigma_1(2, 4, x)$ and $\sigma_2(2, 4, x)$ obtained with maple.

Proof Consider

$$\xi_1(2^m, N, x) = \xi(2^m, N, x - \frac{3}{8}) - \frac{1}{2}$$

and

$$\xi_2(2^m, N, x) = \xi(2^m, N, x - \frac{7}{8}).$$

□

Corollary 3.7 There exists (see Figure 5) $\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that, for all $n, m \in \mathbb{N}$, $[x] \in [-2^n + 1, 2^n]$,

- whenever $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$,

$$|\sigma_1(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m},$$

- and whenever $x \in I_2 = [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$,

$$|\sigma_2(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}.$$

Proof Consider $\sigma_i(2^n, x) = x - \xi_i(2^n, x)$ with the function defined in Corollary 3.6. □

Corollary 3.8 There exists (see Figure 6) $\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that for all $m, n \in \mathbb{N}$, $[x] \in [-2^n + 1, 2^n]$,

- whenever $x \in [\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2}]$,

$$|\lambda(2^m, 2^n, x) - 0| \leq 2^{-m},$$

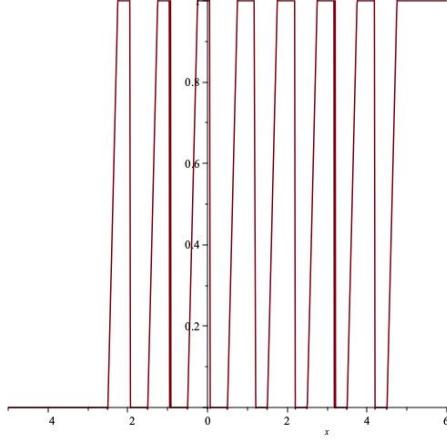


Figure 6: Graphical representation of $\lambda(2, 4, x)$ obtained with maple.

- and whenever $x \in [\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1]$,

$$|\lambda(2^m, 2^n, x) - 1| \leq 2^{-m}.$$

Proof Consider $\lambda(2^m, 2^n, x) = F(\xi(2^{m+1}, 2^n, x - 9/8))$ where

$$F(x) = \mathcal{C}\text{-}\mathfrak{s}(2^{m+1}, 1/4, 1/2, x).$$

□

Corollary 3.9 There exists (see Figure 7) $\text{mod}_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$,

$$|\text{mod}_2(2^m, 2^n, x) - \lfloor x \rfloor \text{ mod } 2| \leq 2^{-m}.$$

Proof We can take

$$\text{mod}_2(2^m, N, x) = 1 - \lambda(2^m, N/2, \frac{1}{2}x + \frac{7}{8})$$

where λ is the function given by Corollary 3.8. □

Corollary 3.10 There exists (see Figure 8) $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$,

$$|\div_2(2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m},$$

where $//$ is the integer division.

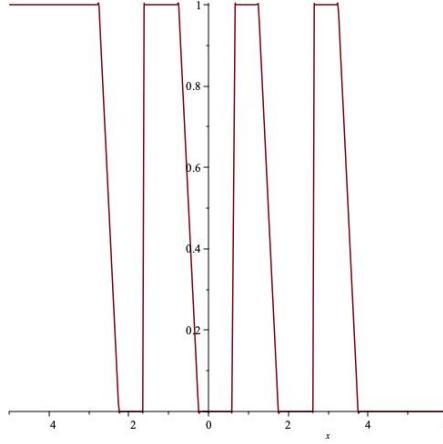


Figure 7: Graphical representation of $\text{mod}_2(2, 4, x)$ obtained with maple.

Proof We can take

$$\div 2(2^m, N, x) = \frac{1}{2}(\sigma_1(2^m, N, x) - \text{mod}_2(2^m, N, x))$$

where mod_2 is the function given by Corollary 3.9, and σ_2 is the function given by Corollary 3.7.

□

3.1 Some properties of sigmoids

Observing that for function

$$\bar{\text{if}}(d, \ell) = 4 \cdot \text{s}(1, 2, 1/2 + d + \ell/4) - 2,$$

for $\ell \in [0, 1]$, we have

$$\begin{aligned} \bar{\text{if}}(0, \ell) &= 0 \\ \bar{\text{if}}(1, \ell) &= \ell. \end{aligned}$$

Applying Lemma 3.4 on this sigmoid, we get:

Lemma 3.11 *There exists $\mathcal{C}\text{-if} \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that, for $\ell \in [0, 1]$,*

- if we take $|d' - 0| \leq 1/4$, then $|\mathcal{C}\text{-if}(2^m, d', \ell) - 0| \leq 2^{-m}$,
- and if we take $|d' - 1| \leq 1/4$, then $|\mathcal{C}\text{-if}(2^m, d', \ell) - \ell| \leq 2^{-m}$.

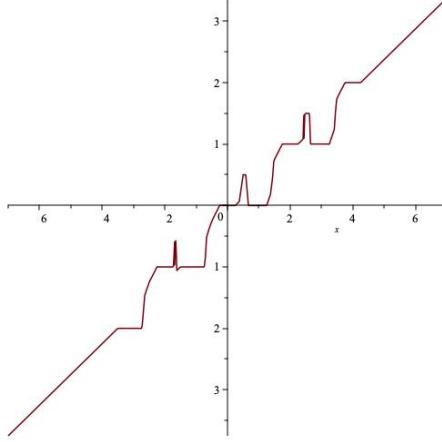


Figure 8: Graphical representation of $\div_2(2, 4, x)$ obtained with maple.

We start by proving the following lemma, which is about some ideal sigmoids:

Lemma 3.12 *For $\ell \in [0, 1]$, for $-1/4 \leq d \leq 1/4$, $\bar{\text{if}}(d, \ell) = 0$, and for $3/4 \leq d \leq 5/4$, $\bar{\text{if}}(d, \ell) = 4(d - 1) + \ell$.*

Consider $\text{if}(d, \ell) = \bar{\text{if}}(\text{s}(1/4, 3/4, x), \ell)$.

- If we take $|d' - 0| \leq 1/4$, then $\text{if}(d', \ell) = 0$,
- and if we take $|d' - 1| \leq 1/4$, then $\text{if}(d', \ell) = \ell$.

Proof Just check that for $d \leq 1/4$, we have $1/2 + d + \ell/4 \leq 1$, and hence $\text{s}(1, 2, 1/2 + d + \ell/4) = 0$, so $\bar{\text{if}}(d, \ell) = 0$. For $3/4 \leq d \leq 5/4$, we have $5/4 \leq 1/2 + d + \ell/4 \leq 2$, and hence $\text{s}(1, 2, 1/2 + d + \ell/4) = d + \ell/4 - 1/2$, and hence $\bar{\text{if}}(1, \ell) = \ell$. The other observation follows. \square

Then we go to versions using \tanh :

Lemma 3.13 *Consider $\bar{\mathcal{C}}\text{-}\bar{\text{if}}(2^m, d, l) = 4\mathcal{C}\text{-}\bar{\text{s}}(2^{m+2}, 1, 2, 1/2 + d + l/4) - 2$. For $\ell \in [0, 1]$, we have $|\bar{\mathcal{C}}\text{-}\bar{\text{if}}(0, \ell) - 0| \leq 2^{-m}$, and $|\bar{\mathcal{C}}\text{-}\bar{\text{if}}(1, \ell) - \ell| \leq 2^{-m}$.*

For $-1/4 \leq d \leq 1/4$, $|\bar{\mathcal{C}}\text{-}\bar{\text{if}}(2^m, d, \ell) - 0| \leq 2^{-m}$, and for $3/4 \leq d \leq 5/4$, $|\bar{\mathcal{C}}\text{-}\bar{\text{if}}(2^m, d, \ell) - 4(d - 1) + \ell| \leq 2^{-m}$.

Consider $\mathcal{C}\text{-}\text{if}(2^m, d, \ell) = \bar{\mathcal{C}}\text{-}\bar{\text{if}}(2^{m+1}, \mathcal{C}\text{-}\bar{\text{s}}(2^{m+1}, 1/4, 3/4, x), \ell)$.

- If we take $|d' - 0| \leq 1/4$, then $|\mathcal{C}\text{-}\text{if}(d', \ell) - 0| \leq 2^{-m}$,

- and if we take $|d' - 1| \leq 1/4$, then $|\mathcal{C}\text{-if}(d', \ell) - \ell| \leq 2^{-m}$.

Proof This is direct from previous lemma and Lemma 3.4. \square

Then Lemma 3.11 follows.

Lemma 3.14 Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some integers, and V_1, V_2, \dots, V_n some constants. There is some function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, that we write $\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$, that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to a real at a distance at most 2^{-m} of V_i , for all $i \in \{1, \dots, n\}$.

We prove the following lemma about some ideal sigmoids: define $\overline{\text{cond}}(x)$ as $\mathfrak{s}(1/4, 3/4, x)$ and $\overline{\mathcal{C}\text{-cond}}(2^m, x)$ as $\mathcal{C}\text{-}\mathfrak{s}(2^m, 1/4, 3/4, x)$.

Lemma 3.15 Assume you are given some integers $\alpha_1, \alpha_2, \dots, \alpha_n$, and some constants V_1, V_2, \dots, V_n . Then there is some function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, written $\text{send}(\alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$, that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to V_i , for all $i \in \{1, \dots, n\}$.

Proof Sort the α_i so that $\alpha_1 < \alpha_2 < \dots, \alpha_n$. Then consider $T_1 + \overline{\text{cond}}(x - \alpha_1)(T_2 - T_1) + \overline{\text{cond}}(x - \alpha_2)(T_3 - T_2) + \dots + \overline{\text{cond}}(x - \alpha_{n-1})(T_n - T_{n-1})$. \square

We now go to versions with \tanh .

Proof of Lemma 3.14 Sort the α_i so that $\alpha_1 < \alpha_2 < \dots, \alpha_n$. Then consider $T_1 + \overline{\mathcal{C}\text{-cond}}(2^{m+c}, x - \alpha_1)(T_2 - T_1) + \overline{\mathcal{C}\text{-cond}}(2^{m+c}, x - \alpha_2)(T_3 - T_2) + \dots + \overline{\mathcal{C}\text{-cond}}(2^{m+c}, x - \alpha_{n-1})(T_n - T_{n-1})$. for some constant c so that $n \max_j (T_j - T_{j+1}) \leq 2^c$. \square

More generally:

Lemma 3.16 Let N be some integer. Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some integers, and $V_{i,j}$ for $1 \leq i \leq n$ some constants, with $0 \leq j < N$. Then there is some function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, that we write $\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$, that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to a real at a distance at most 2^{-m} of $V_{i,j}$, for all $i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}$.

We start by proving the following lemma talking about some ideal sigmoids:

Lemma 3.17 Let N be some integer. Assume some integers $\alpha_1, \alpha_2, \dots, \alpha_n$, and some constants $V_{i,j}$ for $1 \leq i \leq n$, and $0 \leq j < N$ are given. Then there is some function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, that we write $\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$, that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to $V_{i,j}$, for all $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N-1\}$.

Proof If we define the function

$$\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}(x, y)$$

by $\text{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}(Nx + y)$ this works when $x = \alpha_i$ for some i . Considering instead $\text{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}(N \text{send}(\alpha_i \mapsto \alpha_i)_{i \in \{1, \dots, n\}}(x) + y)$ works for any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$.

□

We then go to versions with \tanh :

Proof of Lemma 3.16 If we define the function

$$\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}(x, y)$$

by $\mathcal{C}\text{-send}(2^m, N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}(Nx + y)$ this works when $x = \alpha_i$ for some i . Considering instead

$$\mathcal{C}\text{-send}(2^m, N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}(N \mathcal{C}\text{-send}(2^{m+c}, \alpha_i \mapsto \alpha_i)_{i \in \{1, \dots, n\}}(x) + y)$$

that works for any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$, for some constant c selected as above. □

4 Simulating Turing machines with functions of $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$

This section is devoted to the simulation of a Turing machine using some analytic functions and in particular functions from $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$. We use some ideas from [3] but with several improvements, as we need to deal with errors and avoid multiplications.

Consider without loss of generality some Turing machine $M = (Q, \{0, 1, 3\}, q_{init}, \delta, F)$ using the symbols 0, 1, 3, where $B = 0$ is the blank symbol.

Remark 10 The reason for the choice of symbols 1 and 3 will be made clear later.

We assume $Q = \{0, 1, \dots, |Q| - 1\}$. Let $\dots l_{-k}l_{-k+1} \dots l_{-1}l_0r_0r_1 \dots r_n \dots$ denote the content of the tape of the Turing machine M . In this representation, the head is in front of symbol r_0 , and $l_i, r_i \in \{0, 1, 3\}$ for all i . Such a configuration C can be denoted by $C = (q, l, r)$, where $l, r \in \Sigma^\omega$ are words over alphabet $\Sigma = \{0, 1, 3\}$ and $q \in Q$ denotes the internal state of M . Write: $\gamma_{word} : \Sigma^\omega \rightarrow \mathbb{R}$ for the function that maps a word $w = w_0w_1w_2 \dots$ to the dyadic $\gamma_{word}(w) = \sum_{n \geq 0} w_n 4^{-(n+1)}$.

The idea is that such a configuration C can also be encoded by some element $\bar{C} = (q, \bar{l}, \bar{r}) \in \mathbb{N} \times \mathbb{R}^2$, by considering $\bar{r} = \gamma_{word}(r)$ and $\bar{l} = \gamma_{word}(l)$. In other words, we encode the configuration of a bi-infinite tape Turing machine M by real numbers using their radix 4 encoding, but using only digits 1, 3. Notice that this lives in $Q \times [0, 1]^2$. Denoting the image of $\gamma_{word} : \Sigma^\omega \rightarrow \mathbb{R}$ by \mathcal{I} , this even lives in $Q \times \mathcal{I}^2$.

Remark 11 Notice that \mathcal{I} is a Cantor-like set: it corresponds to the rational numbers that can be written using only 1 and 3 in base 4. We write \mathcal{I}_S for those with at most S digits after the point (i.e. of the form $n/4^S$ for some integer n).

A key point is to observe that

Lemma 4.1 *We can construct some function \overline{Next} in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ that simulates one step of M : given a configuration C , writing C' for the next configuration, we have for all integer m , $\|\overline{Next}(2^m, \bar{C}) - \bar{C}'\| \leq 2^{-m}$.*

Proof We can write $l = l_0l^\bullet$ and $r = r_0r^\bullet$, where l_0 and r_0 are the first letters of l and r , and l^\bullet and r^\bullet corresponding to the (possibly infinite) word $l_{-1}l_{-2} \dots$ and $r_1r_2 \dots$ respectively.

...	l^\bullet	l_0	r_0	r^\bullet	...
$\underbrace{}$			$\underbrace{}$		

The function $Next$ is of the form $Next(q, l, r) = Next(q, l^\bullet l_0, r_0 r^\bullet) = (q', l', r')$ defined as a definition by case:

$$(q', l', r') = \begin{cases} (q', l^\bullet l_0 x, r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \rightarrow) \\ (q', l^\bullet, l_0 x r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases}$$

This provides a first candidate for the function \overline{Next} : Consider the similar function working over the representation of the configurations as reals, considering $r_0 = \lfloor 4\bar{r} \rfloor$

$$\begin{aligned} \overline{Next}(q, \bar{l}, \bar{r}) &= \overline{Next}(q, \overline{l^\bullet l_0}, \overline{r_0 r^\bullet}) = (q', \overline{l'}, \overline{r'}) \\ &= \begin{cases} (q', \overline{l^\bullet l_0 x}, \overline{r^\bullet}) & \text{whenever } \delta(q, \bar{r}_0) = (q', x, \rightarrow) \\ (q', \overline{l^\bullet}, \overline{l_0 x r^\bullet}) & \text{whenever } \delta(q, \bar{r}_0) = (q', x, \leftarrow) \end{cases} \end{aligned}$$

(4–1)

- in the first case “ \rightarrow ”: $\bar{l} = 4^{-1}\bar{l} + 4^{-1}x$ and $\bar{r} = \bar{r}^\bullet = \{4\bar{r}\}$
- in the second case “ \leftarrow ”: $\bar{l} = \bar{r} = \{4\bar{l}\}$ and $\bar{r} = 4^{-2}\{4\bar{r}\} + 4^{-2}x + \lfloor 4\bar{l} \rfloor / 4$

We introduce the following functions: $\rightarrow: Q \times \{0, 1, 3\} \mapsto \{0, 1\}$ and $\leftarrow: Q \times \{0, 1, 3\} \mapsto \{0, 1\}$ such that $\rightarrow(q, a)$ (respectively: $\leftarrow(q, a)$) is 1 when $\delta(q, a) = (\cdot, \cdot, \rightarrow)$ (resp. $(\cdot, \cdot, \leftarrow)$), i.e. the head moves right (resp. left), and 0 otherwise. We define $\text{next}q_a^q = q'$ if $\delta(q, a) = (q', \cdot, \cdot)$, i.e. values (q', x, m) for some x and $m \in \{\leftarrow, \rightarrow\}$.

We can rewrite $\overline{\text{Next}}(q, \bar{l}, \bar{r}) = (q', \bar{l}', \bar{r}')$ as

$$\bar{l}' = \sum_{q, r_0} \left[\rightarrow(q, r_0) \left(\frac{\bar{l}}{4} + \frac{x}{4} \right) + \leftarrow(q, r_0) \{4\bar{l}\} \right]$$

and

$$\bar{r}' = \sum_{q, r_0} \left[\rightarrow(q, r_0) \{4\bar{r}\} + \leftarrow(q, r_0) \left(\frac{\{4r\}}{4^2} + \frac{x}{4^2} + \frac{\lfloor 4\bar{l} \rfloor}{4} \right) \right],$$

and, using notation of Lemma 3.16, $q' = \text{send}((q, r) \mapsto \text{next}q_r^q)_{q \in Q, r \in \{0, 1, 3\}}(q, \lfloor 4\bar{r} \rfloor)$.

Our problem with such expressions is that they involve some discontinuous functions such as the integer part and the fractional part function, and we would rather have analytic (hence continuous) functions. However, a key point is that from our trick of using only symbols 1 and 3, we are sure that in an expression like $\lfloor 4\bar{r} \rfloor$, either it values 0 (this is the specific case where there remain only blanks in r), or that $4\bar{r}$ lives in an interval $[1, 2]$ or in interval $[3, 4]$.

That means that we could replace $\lfloor 4\bar{r} \rfloor$ by $\sigma(4\bar{r})$ if we take σ as some continuous function that would be affine and values respectively 0, 1 and 3 on $\{0\} \cup [1, 2] \cup [3, 4]$ (that is to say matches $\lfloor 4\bar{r} \rfloor$ on this domain). A possible candidate is $\sigma(x) = \mathfrak{s}(1/4, 3/4, x) + \mathfrak{s}(9/4, 11/4, x)$. Then considering $\xi(x) = x - \sigma(x)$, then $\xi(4\bar{r})$ would be the same as $\{4\bar{r}\}$: that is, considering $r_0 = \sigma(4\bar{r})$, replacing in the above expression every $\{4\cdot\}$ by $\xi(\cdot)$, and every $\lfloor \cdot \rfloor$ by $\sigma(\cdot)$, and get something that would still work the same, but using only continuous functions.

But, we would like to go to some analytic functions and not only continuous functions, and it is well-known that an analytic function that equals some affine function on some interval (e.g. on $[1, 2]$) must be affine, and hence cannot be 3 on $[3, 4]$. But the point is that we can try to tolerate errors, and replace $\mathfrak{s}(\cdot, \cdot)$ by $\mathcal{C}\text{-}\mathfrak{s}(2^{m+c}, \cdot, \cdot)$ in the expressions above for σ and ξ , taking c such that $(3 + 1/4^2)3|Q| \leq 2^c$. This would just introduce some error at most $(3 + 1/4^2)3|Q|2^{-c}2^{-m} \leq 2^{-m}$.

Remark 12 We could also replace every $\rightarrow (q, r)$ in above expressions for \bar{l}' and \bar{r}' by $\mathcal{C}\text{-send}(k, (q, r) \mapsto \rightarrow (q, r))(q, \sigma(4\bar{r}))$, for a suitable error bound k , and symmetrically for $\leftarrow (q, r)$. However, if we do so, we still might have some multiplications in the above expressions.

The key is to use Lemma 3.11: we can also write the above expressions as

$$\begin{aligned} \bar{l}' &= \sum_{q,r} \left[\mathcal{C}\text{-if} \left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \rightarrow (q, r))(q, \sigma(4\bar{r})), \frac{\bar{l}}{4} + \frac{x}{4} \right) \right. \\ &\quad \left. + \mathcal{C}\text{-if} \left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \leftarrow (q, r))(q, \sigma(4\bar{r})), \xi(4\bar{l}) \right) \right] \\ \bar{r}' &= \sum_{q,r} \left[\mathcal{C}\text{-if} \left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \rightarrow (q, r))(q, \sigma(4\bar{r})), \xi(4\bar{r}) \right) \right. \\ &\quad \left. + \mathcal{C}\text{-if} \left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \leftarrow (q, r))(q, \sigma(4\bar{r})), \frac{\xi(4r)}{4^2} + \frac{x}{4^2} + \frac{\sigma(4\bar{l})}{4} \right) \right] \end{aligned}$$

and still have the same bound on the error. \square

Once we have one step, we would like to simulate some arbitrary computation of a Turing machine, by considering the iterations of function *Next*.

The problem with the above construction is that even if we start from the exact encoding \bar{C} of a configuration, it introduces some error (even if at most 2^{-m}). If we want to apply again the function *Next*, then we will start not exactly from the encoding of a configuration. Looking at the choice of the function σ , a small error can be tolerated (roughly if the process does not involve points at a distance less than $1/4$ of \mathcal{I}), but this error is amplified (roughly multiplied by 4 on some component), before introducing some new errors (even if at most 2^{-m}). The point is that if we repeat the process, very soon it will be amplified, up to a level where we have no true idea or control about what becomes the value of the above function.

However, if we know some bound on the space used by the Turing machine, we can correct it to get at most some fixed additive error: a Turing machine using a space S uses at most S cells to the right and to the left of the initial position of its head. Consequently, a configuration $C = (q, l, r)$ of such a machine involves words l and r of length at most S . Their encoding \bar{l} , and \bar{r} are expected to remain in \mathcal{I}_{S+1} . Consider $\text{round}_{S+1}(\bar{l}) = \lfloor 4^{S+1}\bar{l} \rfloor / 4^{S+1}$. For a point \bar{l} of \mathcal{I}_{S+1} , $4^{S+1}\bar{l}$ is an integer, and $\bar{l} = \text{round}_{S+1}(\bar{l})$. But now, for a point $\tilde{\bar{l}}$ at a distance less than $4^{-(S+2)}$ from a point $\bar{l} \in \mathcal{I}_{S+1}$, $\text{round}_{S+1}(\tilde{\bar{l}}) = \bar{l}$. In other words, round_{S+1} “deletes” errors of order $4^{-(S+2)}$. Consequently, we can replace every \bar{l} in the above expressions by $\sigma_1(2^{2S+4}, 2^{2S+3}, 4^{S+1}\bar{l})/4^{S+1}$, as this is close to $\text{round}_{S+1}(\bar{l})$, and the same for \bar{r} , where σ_1 is the function from Corollary 3.7. We could also replace m by $m + 2S + 4$ to guarantee that $2^{-m} \leq 4^{-(S+2)}$. We get the following important improvement of the previous lemma:

Lemma 4.2 We can construct some function \overline{Next} in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ that simulates one step of M , i.e. that computes the $Next$ function sending a configuration \overline{C} of Turing machine M to \overline{C}' , where C' is the next one: $\|\overline{Next}(2^m, 2^S, \overline{C}) - \overline{C}'\| \leq 2^{-m}$.

Furthermore, it is robust to errors on its input, up to space S : considering $\|\tilde{C} - \overline{C}\| \leq 4^{-(S+2)}$, $\|\overline{Next}(2^m, 2^S, \tilde{C}) - \overline{C}'\| \leq 2^{-m}$ remains true.

Proposition 4.3 Consider some Turing machine M that computes some function $f : \Sigma^* \rightarrow \Sigma^*$ in some time $T(\ell(\omega))$ on input ω . One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$ in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ that does the same: $\tilde{f}(2^m, 2^{T(\ell(\omega))}, \gamma_{word}(\omega))$ that is at most 2^{-m} far from $\gamma_{word}(f(\omega))$.

Proof The idea is to define the function \overline{Exec} that maps some time 2^t and some initial configuration C to the configuration at time t . This can be obtained using previous lemmas by

$$\begin{cases} \overline{Exec}(2^m, 0, 2^T, C) &= C \\ \overline{Exec}(2^m, 2^{t+1}, 2^T, C) &= \overline{Next}(2^m, 2^T, \overline{Exec}(2^m, 2^t, 2^T, C)). \end{cases}$$

We can then get the value of the computation as $\overline{Exec}(2^m, 2^{T(\ell(\omega))}, 2^{T(\ell(\omega))}, C_{init})$ on input ω , considering $C_{init} = (q_0, 0, \gamma_{word}(\omega))$. By applying some projection, we get the following function $\tilde{f}(2^m, 2^T, y) = \pi_3^3(\overline{Exec}(2^m, 2^T, 2^T, (q_0, 0, y)))$ that satisfies the property. \square

Actually, in order to get FPSPACE, observe that we can also replace the linear length ODE by a linear ODE.

Proposition 4.4 Consider some Turing machine M that computes some function $f : \Sigma^* \rightarrow \Sigma^*$ in some polynomial space $S(\ell(\omega))$ on input ω . One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$ in $\mathbb{R}\mathbb{L}\mathbb{D}^\circ$ that does the same: we have $\tilde{f}(2^m, 2^{S(\ell(\omega))}, \gamma_{word}(\omega))$ that is at most 2^{-m} far from $\gamma_{word}(f(\omega))$.

Proof The idea is the same, but not working with powers of 2, and with linear ODE: define the function \overline{Exec} that maps some time t and some initial configuration C to the configuration at time t . This can be obtained the using previous lemma by

$$\begin{cases} \overline{Exec}(2^m, 0, 2^S, C) &= C \\ \overline{Exec}(2^m, t + 1, 2^S, C) &= \overline{Next}(2^m, 2^S, \overline{Exec}(2^m, t, 2^S, C)). \end{cases}$$

In order to claim this is a robust linear ODE, we need to state that $\overline{Exec}(2^m, t, 2^S, C)$ is polynomially numerically stable: but this holds, since to estimate this value at 2^{-n} it is sufficient to work at precision $4^{-\max(m,n,S+2)}$ (independently of t , from the rounding).

We can then get the value of the computation as $\overline{Exec}(2^m, 2^{S(\ell(\omega))}, 2^{S(\ell(\omega))}, C_{init})$ on input ω , considering $C_{init} = (q_0, 0, \gamma_{word}(\omega))$. By applying some projection, we get the following function $\tilde{f}(2^m, 2^S, y) = \pi_3^3(\overline{Exec}(2^m, S, 2^S, (q_0, 0, y)))$ that satisfies the property. \square

5 Converting integers and dyadics to words, and conversely

One point of our simulations of Turing machines is that they work over \mathcal{I} , through encoding γ_{word} , while we would like to talk about integers and real numbers: we need to be able to convert an integer (more generally a dyadic) into some encoding over \mathcal{I} and conversely.

Fix the following encoding: every digit in the binary expansion of d is encoded by a pair of symbols in the radix 4 expansion of $\bar{d} \in \mathcal{I} \cap [0, 1]$: digit 0 (respectively: 1) is encoded by 11 (resp. 13) if before the “decimal” point in d , and digit 0 (respectively: 1) is encoded by 31 (resp. 33) if after. For example, for $d = 101.1$ in base 2, $\bar{d} = 0.13111333$ in base 4.

Lemma 5.1 (From \mathbb{N} to \mathcal{I}) *We can construct some function $Decode : \mathbb{N}^2 \rightarrow \mathbb{R}$ in \mathbb{LDD}° that maps m and n to some point at distance less than 2^{-m} from $\gamma_{word}(\bar{n})$.*

Proof Recall the functions provided by Corollaries 3.9 and 3.10. The idea is to iterate $\ell(n)$ times function

$$F(\bar{r}_1, \bar{l}_2) = \begin{cases} (\div_2(\bar{r}_1), (\bar{l}_2 + 5)/4) & \text{whenever } \text{mod}_2(\bar{r}_1) = 0 \\ (\div_2(\bar{r}_1), (\bar{l}_2 + 7)/4) & \text{whenever } \text{mod}_2(\bar{r}_1) = 1. \end{cases}$$

over $(n, 0)$, and then projects on the second argument.

This can be done in \mathbb{LDD}° by considering $F(2^m, 2^M, \bar{r}_1, \bar{l}_2) = \mathcal{C}\text{-send}(2^{m+1}, 0 \mapsto (\div_2(2^{m+1}, 2^M, \bar{r}_1), (\bar{l}_2 + 5)/4), 1 \mapsto (\div_2(2^{m+1}, 2^M, \bar{r}_1), (\bar{l}_2 + 7)/4))((\bar{r}_1), \text{and then}$

$$Decode(2^m, n) = \pi_2^2(G(2^{m+\ell(n)}, 2^{\ell(n)+1}, 2^{\ell(n)}, n, 0)),$$

with

$$\begin{cases} G(2^m, 2^M, 2^l, 2^0, n, 0) &= (n, 0) \\ G(2^m, 2^M, 2^{l+1}, r, l) &= F(2^m, 2^M, G(2^m, 2^l, r, l)). \end{cases}$$

The global error will be at most $2^{-m-\ell(n)} \times \ell(n) \leq 2^{-m}$. \square

This technique can be extended to consider decoding of tuples: there is a function $Decode : \mathbb{N}^{d+1} \rightarrow \mathbb{R}$ in \mathbb{LDL}° that maps m and \mathbf{n} to some point at distance less than 2^{-m} from $\gamma_{word}(\bar{\mathbf{n}})$, with $\bar{\mathbf{n}}$ defined componentwise.

Conversely, given \bar{d} , we need a way to construct d . Actually, as we will need to avoid multiplications, we state that we can even do something stronger: given \bar{d} , and (some bounded) λ we can construct λd .

Lemma 5.2 (From \mathcal{I} to \mathbb{R} , and multiplying in parallel) We can construct some function $EncodeMul : \mathbb{N}^2 \times [0, 1] \times \mathbb{R} \rightarrow \mathbb{R}$ in \mathbb{LDL}° that maps $m, 2^S, \gamma_{word}(\bar{d})$ and (bounded) λ to some real at distance at most 2^{-m} from λd , whenever \bar{d} is of length less than S .

Proof The idea is to do as in the proof of previous lemma, but considering

$$F(\bar{r}_1, \bar{l}_2, \lambda) = \begin{cases} (\sigma(16\bar{r}_1), 2\bar{l}_2 + 0, \lambda) & \text{whenever } i(16\bar{r}_1) = 5 \\ (\sigma(16\bar{r}_1), 2\bar{l}_2 + \lambda, \lambda) & \text{whenever } i(16\bar{r}_1) = 7 \\ (\sigma(16\bar{r}_1), (\bar{l}_2 + 0)/2, \lambda) & \text{whenever } i(16\bar{r}_1) = 13 \\ (\sigma(16\bar{r}_1), (\bar{l}_2 + \lambda)/2, \lambda) & \text{whenever } i(16\bar{r}_1) = 15 \end{cases}$$

iterated S times over suitable approximation of the rounding $\text{round}_{S+1}(\gamma_{word}(\bar{d}), 0, \lambda))$, with σ and ξ constructed as approximation of the integer and fractional part, as before. \square

6 Proofs and applications

Theorem 1.3 follows from point 1. of next Proposition for one inclusion, and previous simulation of Turing machines for the other.

We start by stating the following (similar to the statement about \mathbb{LDL}^\bullet in [3]):

Proposition 6.1 (1) All functions of \mathbb{LDL}° are computable (in the sense of computable analysis) in polynomial time.

- (2) All functions of \mathbb{RLD} are computable (in the sense of computable analysis) in polynomial space.

Proof The proof consists in observing this holds for the basic functions and that composition preserves polynomial time (respectively: space) computability and also by linear length ODEs. This latter fact is established by computable analysis arguments, reasoning on some explicit formula giving the solution of linear length ODE. The proof is similar to the statement about \mathbb{LDL}^\bullet in [3]. In order to be self-content, we repeat in Appendix C, how this is established in [3].

Regarding space, the main issue is the need to prove the schema given by Definition 2.7 guarantees \mathbf{f} is in **FPSpace** when \mathbf{u} , \mathbf{g} , and \mathbf{h} are. Assuming condition 1. of Definition 2.7 would not be sufficient: the problem is that $\mathbf{f}(x, \mathbf{y})$ may polynomially grow too fast or have a modulus function that would grow too fast. The point is, in Definition 2.7, we assumed \mathbf{f} to be both bounded and satisfying 2., i.e. polynomial numerical robustness. With these hypotheses, it is sufficient to work with the precision given by this robustness condition and these conditions guarantee the validity of computing with such approximated values. \square

We now go to various applications of the proposition and of our toolbox. First, we state a characterisation of **FPTIME** for general functions, covering both the case of a function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d''}$, $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d''}$ as a special case: only the first type (sequences) was covered by [3].

Theorem 6.2 (Theorem 1.4) A function $\mathbf{f} : \mathbb{R}^d \times \mathbb{N}^{d''} \rightarrow \mathbb{R}^{d''}$ is computable in polynomial time iff there exists $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''+2} \rightarrow \mathbb{R}^{d''} \in \mathbb{LDL}^\circ$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$.

The reverse implication of Theorem 6.2 follows from Proposition 6.1, (1.) and arguments from computable analysis.

Proof of reverse implication of Theorem 6.2 Assume there exists $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''+2} \rightarrow \mathbb{R}^{d''} \in \mathbb{LDL}^\circ$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$.

From Proposition 6.1, (1.), we know that $\tilde{\mathbf{f}}$ is computable in polynomial time (in the binary length of its arguments). Then $\mathbf{f}(\mathbf{x}, \mathbf{m})$ is computable: indeed, given \mathbf{x} , \mathbf{m} and n , we can approximate $\mathbf{f}(\mathbf{x}, \mathbf{m})$ at precision 2^{-n} on $[-2^X, 2^X]$ as follows: approximate

$\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1})$ at precision $2^{-(n+1)}$ by some rational q , and output q . We will then have

$$\begin{aligned}\|q - \mathbf{f}(\mathbf{x}, \mathbf{m})\| &\leq \|q - \tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1})\| + \|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1}) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \\ &\leq 2^{-(n+1)} + 2^{-(n+1)} \\ &\leq 2^{-n}.\end{aligned}$$

All of this is done in polynomial time in n and the size of \mathbf{m} , and hence we get that \mathbf{f} is polynomial time computable from definitions. \square

For the direct implication, for sequences, that is to say, functions of type $\mathbf{f} : \mathbb{N}^{d''} \rightarrow \mathbb{R}^{d'}$ (i.e. $d = 0$, the case considered in [3]) we are almost done: reasoning componentwise, we only need to consider $f : \mathbb{N}^{d''} \rightarrow \mathbb{R}$ (i.e. $d' = 1$). As the function is polynomial time computable, this means that there is a polynomial time computable function $g : \mathbb{N}^{d''+1} \rightarrow \{1, 3\}^*$ so that on $\mathbf{m}, 2^n$, it provides the encoding $\overline{\phi(\mathbf{m}, n)}$ of some dyadic $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ for all \mathbf{m} . The problem is then to decode, compute and encode the result to produce this dyadic, using our previous toolbox.

More precisely, from Proposition 4.3, we get \tilde{g} with

$$|\tilde{g}(2^e, 2^{p(\max(\mathbf{m}, n))}, \text{Decode}(2^e, \mathbf{m}, n)) - \gamma_{\text{word}}(g(\mathbf{m}, n))| \leq 2^{-e}$$

for some polynomial p corresponding to the time required to compute g , and $e = \max(p(\max(\mathbf{m}, n)), n)$. Then we need to transform the value to the correct dyadic: we mean

$$\tilde{\mathbf{f}}(\mathbf{m}, n) = \text{EncodeMul}(2^e, 2^t, \tilde{g}(2^e, 2^t, \text{Decode}(2^e, \mathbf{m}, n)), 1),$$

where $t = p(\max(\mathbf{m}, n))$, $e = \max(p(\max(\mathbf{m}, n)), n)$ provides a solution such that $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$.

Remark 13 This is basically what is done in [3], except that we do it here with analytic functions. However, as already observed in [3], this cannot be done for the case $d \geq 1$, i.e. for example for $f : \mathbb{R} \rightarrow \mathbb{R}$. The problem is that we used the fact that we can decode: *Decode* maps an integer n to its encoding \bar{n} (but is not guaranteed to do something valid on non-integers). There cannot exist such functions that would be valid over all reals, as such functions must be continuous, and there is no way to map continuously real numbers to finite words. This is where the approach of the article [3] is stuck.

To solve this, we use an adaptive barycentric technique. For simplicity, and pedagogy, we discuss only the case of a polynomial time computable function $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$.

From standard arguments from computable analysis (see e.g. [Corollary 2.21][20]), the following holds and the point is to be able to realise all this with functions from \mathbb{LDL}° .

Lemma 6.3 *Assume $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$ is computable in polynomial time. There exists some polynomial $m : \mathbb{N}^2 \rightarrow \mathbb{N}$ and some $\tilde{f} : \mathbb{N}^4 \rightarrow \mathbb{Z}$ computable in polynomial time such that for all $x \in \mathbb{R}$, $\|2^{-n}\tilde{f}(\lfloor 2^{m(n,M)}x \rfloor, u, 2^M, 2^n) - f(x, u)\| \leq 2^{-n}$ whenever $\frac{x}{2^{m(n,M)}} \in [-2^M, 2^M]$.*

Assume we consider an approximation σ_i (with either $i = 1$ or $i = 2$) of the integer part function given by Lemma 3.7. Then, given n, M , when $2^{m(n,M)}x$ falls in some suitable interval I_i for σ_i (see the statement of Lemma 3.7), we are sure that $\sigma_i(2^e, 2^{m(n,M)+X+1}, 2^{m(n,M)}x)$ is at some distance upon control from $\lfloor 2^{m(n,M)}x \rfloor$. Consequently, $2^{-n}\tilde{f}(\sigma_i(2^{m(n,M)+X+1}, 2^{m(n,M)}x), u, 2^M, 2^n)$ provides some 2^{-n} -approximation of $f(x, u)$, up to some error upon control. When this holds, we then use an argument similar to what we describe for sequences: using functions from \mathbb{LDL}° , we can decode, compute, and encode the result to provide this dyadic. It is provided by an expression $Formula_i(x, u, M, n)$ of the form $EncodeMul(2^e, 2^t, \tilde{f}(2^2, 2^t, Decode(2^e, \sigma_i(2^e, 2^M, 2^{m(n,M)}x))), 2^{-n})$.

The problem is that it might also be the case that $2^{m(n,M)}x$ falls in the complement of the intervals $(I_i)_i$. In that case, we have no clear idea of what could be the value of $\sigma_i(2^e, 2^{m(n,M)+X+1}, 2^{m(n,M)}x)$, and consequently of what might be the value of the above expression $Formula_i(x, u, M, n)$. But the point is that when it happens for an x for σ_1 , we could have used σ_2 , and this would work, as one can check that the intervals of type I_1 cover the complements of the intervals of type I_2 and conversely. They also overlap, but when x is both in some I_1 and I_2 , $Formula_1(x, u, M, n)$ and $Formula_2(x, u, M, n)$ may differ, but they are both 2^{-n} approximations of $f(x)$.

The key is to compute some suitable "adaptive" barycenter, using function λ , provided by Corollary 3.8. Writing \approx for the fact that two values are closed up to some controlled bounded error, observe from the statements of Corollary 3.8 and 3.7

- that whenever $\lambda(\cdot, 2^n, x) \approx 0$, we know that $\sigma_2(\cdot, 2^n, x) \approx \lfloor x \rfloor$;
- that whenever $\lambda(\cdot, 2^n, x) \approx 1$ we know that $\sigma_1(\cdot, 2^n, x) \approx \lfloor x \rfloor$;
- that whenever $\lambda(\cdot, 2^n, x) \in (0, 1)$, we know that $\sigma_1(\cdot, 2^n, x) \approx \lfloor x \rfloor + 1$ and $\sigma_2(\cdot, 2^n, x) \approx \lfloor x \rfloor$.

That means that if we consider

$$\lambda(\cdot, 2^n, x)Formula_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, n))Formula_2(x, u, M, n)$$

we are sure to be close (up to some bounded error) to some 2^{-n} approximation of $f(x)$. There remains that this requires some multiplication with λ . But from the form of $Formula_i(x, u, M, n)$, this could be also be written as follows, ending the proof of Theorem 6.2.

(6–1)

$$\begin{aligned} & EncodeMul(2^e, 2^t, \tilde{f}(2^e, 2^t, Decode(2^e, \sigma_1(2^e, 2^M, 2^{m(n,M)}x))), \lambda(2^e, 2^M, 2^{m(n,M)}x)2^{-n}) + \\ & EncodeMul(2^e, 2^t, \tilde{f}(2^e, 2^t, Decode(2^e, \sigma_2(2^e, 2^M, 2^{m(n,M)}x))), (1 - \lambda(2^e, 2^M, 2^{m(n,M)}x))2^{-n}) \end{aligned}$$

Proof of Theorem 1.3 We know that a function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ from $\overline{\text{LDL}}^\circ$ is polynomial time computable by Proposition 6.1, (1.). That means that we can approximate it with arbitrary precision, in particular precision $\frac{1}{4}$ in polynomial time. Given such an approximation \mathbf{q} , if we know it is some integer, it is easy to determine which integer it is: return (componentwise) the closest integer to \mathbf{q} .

Conversely, if we have a function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{N}^{d'}$ that is polynomial time computable, our previous simulations of Turing machines provide a function in $\overline{\text{LDL}}^\circ$ that computes it at any required precision, in particular $1/4$. \square

From the fact that we have the reverse direction in Theorem 6.2, it is natural to consider the operation that maps $\tilde{\mathbf{f}}$ to \mathbf{f} . Namely, we introduce the operation *ELim* (this stands for Effective Limit):

Definition 6.4 (Operation *ELim*) Given $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''} \times \mathbb{N} \rightarrow \mathbb{R}^{d'} \in \overline{\text{LDL}}^\circ$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$, then *ELim*($\tilde{\mathbf{f}}$) is the (clearly uniquely defined) corresponding function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$.

Theorem 6.5 A continuous function \mathbf{f} is computable in polynomial time if and only if all its components belong to $\overline{\text{LDL}}^\circ$, where

$$\overline{\text{LDL}}^\circ = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \overline{\text{cond}}(x), \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE, } \text{ELim}].$$

For the reverse direction, by induction, the only thing to prove is that the class of functions from integers computable in polynomial time is preserved by the operation *ELim*. Take such a function $\tilde{\mathbf{f}}$. By definition, given $\mathbf{x}, \mathbf{m}, X$ we can compute $\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n)$ with precision 2^{-n} in time polynomial in n . This must be, by definition of *ELim* schema, some approximation of $\mathbf{f}(\mathbf{x}, \mathbf{m})$ over $[-2^X, 2^X]$, and hence \mathbf{f} is computable in polynomial time. This also gives directly Theorem 1.4 as a corollary.

From the proofs, we also get a normal form theorem, namely formula (6–1). In particular,

Theorem 6.6 Any function $f : \mathbb{N}^d \times \mathbb{R}^{d''} \rightarrow \mathbb{R}^{d'}$ can be obtained from the class $\overline{\text{LDL}}^\circ$ using only one schema *ELim*.

We obtain the statements for polynomial space computability (Theorems 1.5 and 1.6) replacing LDL° by RLD° , using similar reasoning about space instead of time, considering point 2. instead of 1. of Proposition 6.1, and Proposition 4.4 instead of Proposition 4.3. We now comment on the relations with formal neural network models. In this article, we are programming with tanh and sigmoids. We expressed the sigmoids in terms of the ReLU function, through Lemma 3.1. Function tanh could be replaced by arctan: the key was to be able to approximate the ReLU function with tanh (Lemma 3.1) and this can be proved to hold also for arctan, using error bounds on arctan established in [18].

Now, given some function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, some error and some time t , our expressions provide explicit expressions in LDL° of an approximation of what is computed by a Turing machine at time t uniformly over any compact domain.

Remark 14 The formula 6–1 can be seen as a function that generates uniformly a family of circuits/formal \mathcal{C} - \mathfrak{s} approximating a given function at some given precision over some given domain. The functions we generate are the composition of essentially linear functions, which can be considered as layers of formal neural networks¹.

References

- [1] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.
- [2] Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. Submitted. Journal version of [3]. Preliminary version available on <https://arxiv.org/abs/2209.13599>.
- [3] Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality - 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 - September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2022. MCU’22 Best Student Paper Award. doi: [10.1007/978-3-031-13502-6\4](https://doi.org/10.1007/978-3-031-13502-6_4).

¹With a concept of neural network that is not assuming that the last layer of the network is made of neurons, and that result may be outputted by some linear combination of the neurons in the last layer.

- [4] Manon Blanc and Olivier Bournez. A characterisation of functions computable in polynomial time and space over the reals with discrete ordinary differential equationsimulation of turing machines with analytic discrete odes. In *Mathematical Foundations of Computer Science (MFCS'2023)*, 2023.
- [5] Manon Blanc and Olivier Bournez. Measuring robustness of dynamical systems. relating time and space to length and precision, 2023. URL: <https://arxiv.org/abs/2301.12723>, doi:10.48550/ARXIV.2301.12723.
- [6] Olivier Bournez. Programming with ordinary differential equations: Some first steps towards a programming language. In Ulrich Berger, Johanna N. Y. Franklin, Florin Manea, and Arno Pauly, editors, *Revolutions and Revelations in Computability - 18th Conference on Computability in Europe, CiE 2022, Swansea, UK, July 11-15, 2022, Proceedings*, volume 13359 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2022. doi:10.1007/978-3-031-08740-0_4.
- [7] Olivier Bournez, Manuel L. Campagnolo, Daniel Graça, and Emmanuel S. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.
- [8] Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th Int Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPICS*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [9] Olivier Bournez and Arnaud Durand. A characterization of functions over the integers computable in polynomial time using discrete ordinary differential equations. *Computational Complexity*, 32(2):7, 2023.
- [10] Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In *International Colloquium on Automata Language Programming, ICALP'2017*, 2017.
- [11] Olivier Bournez and Amaury Pouly. A survey on analog models of computation. In Vasco Brattka and Peter Hertling, editors, *Handbook of Computability and Complexity in Analysis*. Springer, 2020.
- [12] Vasco Brattka. Recursive characterization of computable real-valued functions and relations. *Theoretical Computer Science*, 162(1):45–77, 1996.
- [13] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In *New computational paradigms*, pages 425–491. Springer, 2008.
- [14] P. Clote. Computational models and function algebras. In Edward R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. North-Holland, Amsterdam, 1998.
- [15] Peter Clote and Evangelos Kranakis. *Boolean functions and computation models*. Springer Science & Business Media, 2013.
- [16] Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

- [17] Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Computational Methods in Systems Biology-CMSB 2017*, 2017. CMSB'2017 Best Paper Award.
- [18] Daniel S. Graça. *Computability with Polynomial Differential Equations*. PhD thesis, Instituto Superior Técnico, 2007.
- [19] Daniel S. Graça and Ning Zhong. *Handbook of Computability and Complexity in Analysis*, chapter Computability of Differential Equations. Springer., 2018.
- [20] Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.
- [21] D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, number 960 in Lecture Notes in Computer Science, pages 177–194, 1995.
- [22] Daniel Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.
- [23] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of Poly-Time. *Fundamenta Informatica*, 19(1,2):167,184, 1993.
- [24] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380, Kazimierz, Poland, 1995. Springer.
- [25] Keng Meng Ng, Nazanin R Tavana, and Yue Yang. A recursion theoretic foundation of computation over real numbers. *Journal of Logic and Computation*, 31(7):1660–1689, 2021.
- [26] Marian Boykan Pour-El and J. Ian Richards. A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17:61–90, 1979.
- [27] Hava T. Siegelmann. *Neural Networks and Analog Computation - Beyond the Turing Limit*. Birkäuser, 1999.
- [28] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, February 1995.
- [29] David B Thompson. Subrecursiveness: Machine-independent notions of computability in restricted time and storage. *Mathematical Systems Theory*, 6(1-2):3–15, 1972.
- [30] Klaus Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.

A Concepts from computable analysis

When we say that a function $f : S_1 \times \dots \times S_d \rightarrow \mathbb{R}^{d'}$ is (respectively: polynomial-time) computable this will always be in the sense of computable analysis: see e.g. [13, 30]. We recall here the basic concepts and definitions, mostly following the book [20], whose subject is complexity theory in computable analysis. This section is repeating the formalisation proposed in [3] done to mix complexity issues dealing with integer and real arguments: a dyadic number d is a rational number with a finite binary expansion. That is to say $d = m/2^n$ for some integers $m \in \mathbb{Z}$, $n \in \mathbb{N}$, $n \geq 0$. Let \mathbb{D} be the set of all dyadic rational numbers. We denote by \mathbb{D}_n the set of all dyadic rationals d with a representation s of precision $\text{prec}(s) = n$; that is, $\mathbb{D}_n = \{m \cdot 2^{-n} \mid m \in \mathbb{Z}\}$.

Definition A.1 ([20]) For each real number x , a function $\phi : \mathbb{N} \rightarrow \mathbb{D}$ is said to binary converge to x if for all $n \in \mathbb{N}$, $\text{prec}(\phi(n)) = n$ and $|\phi(n) - x| \leq 2^{-n}$. Let CF_x (Cauchy function) denotes the set of all functions binary converging to x .

Intuitively, a Turing machine M computes a real function f the following way: 1. The input x to f , represented by some $\phi \in CF_x$, is given to M as an oracle; 2. The output precision 2^{-n} is given in the form of integer n as the input to M ; 3. The computation of M usually takes two steps, though sometimes these two steps may be repeated an indefinite number of times; 4. M computes, from the output precision 2^{-n} , the required input precision 2^{-m} ; 5. M queries the oracle to get $\phi(m)$, such that $|\phi(m) - x| \leq 2^{-m}$, and computes from $\phi(m)$ an output $d \in \mathbb{D}$ with $\|d - f(x)\| \leq 2^{-n}$. More formally:

Definition A.2 ([20]) A real function $f : \mathbb{R} \rightarrow \mathbb{R}$ is computable if there is a function-oracle TM M such that for each $x \in \mathbb{R}$ and each $\phi \in CF_x$, the function ψ computed by M with oracle ϕ (i.e., $\psi(n) = M^\phi(n)$) is in $CF_{f(x)}$. We say the function f is computable on the interval $[a, b]$ if the above condition holds for all $x \in [a, b]$.

Remark 15 Given some $x \in \mathbb{R}$, such an oracle TM M can determine some integer X such that $x \in [-2^X, 2^X]$.

A.1 On computable analysis: Complexity

Assume that M is an oracle machine computing f on a domain G . For any oracle $\phi \in CF_x$, with $x \in G$, let $T_M(\phi, n)$ be the number of steps for M to halt on input n with oracle ϕ , and $T'_M(x, n) = \max \{T_M(\phi, n) \mid \phi \in CF_x\}$. The time complexity of f is defined as follows:

Definition A.3 ([20]) Let G be bounded closed interval $[a, b]$. Let $f : G \rightarrow \mathbb{R}$ be a computable function. Then, we say that the time complexity of f on G is bounded by a function $t : G \times \mathbb{N} \rightarrow \mathbb{N}$ if there exists an oracle TM M which computes f such that for all $x \in G$ and all $n > 0$, $T'_M(x, n) \leq t(x, n)$.

In other words, the idea is to measure the time complexity of a real function based on two parameters: input real number x and output precision 2^{-n} . Sometimes, it is more convenient to simplify the complexity measure to be based on only one parameter, the output precision. For this purpose, we say the uniform time complexity of f on G is bounded by a function $t' : \mathbb{N} \rightarrow \mathbb{N}$ if the time complexity of f on G is bounded by a function $t : G \times \mathbb{N} \rightarrow \mathbb{N}$ with the property that for all $x \in G$, $t(x, n) \leq t'(n)$.

However, if we do so, it is important to realise that if we had taken $G = \mathbb{R}$ in the previous definition, for unbounded functions f , the uniform time complexity would not have existed, because the number of moves required to write down the integral part of $f(x)$ grows as x approaches $+\infty$ or $-\infty$. Therefore, the approach of [20] is to do as follows (The bounds -2^X and 2^X are somewhat arbitrary, but are chosen here because the binary expansion of any $x \in (-2^n, 2^n)$ has at most n bits in the integral part).

Definition A.4 (Adapted from [20]) For functions $f(x)$ whose domain is \mathbb{R} , we say that the (non-uniform) time complexity of f is bounded by a function $t' : \mathbb{N}^2 \rightarrow \mathbb{N}$ if the time complexity of f on $[-2^X, 2^X]$ is bounded by a function $t : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that $t(x, n) \leq t'(X, n)$ for all $x \in [-2^X, 2^X]$.

Remark 16 The space complexity of a real function is defined similarly. We say the space complexity of $f : G \rightarrow \mathbb{R}$ is bounded by a function $s : G \times \mathbb{N} \rightarrow \mathbb{N}$ if there is an oracle TM M which computes f such that for any input n and any oracle $\phi \in CF_x, M^\phi(n)$ uses $\leq s(x, n)$ cells, and the uniform space complexity of f is bounded by $s' : \mathbb{N} \rightarrow \mathbb{N}$ if for all $x \in G$ and all $\phi \in CF_x, M^\phi(n)$ uses $\leq s'(n)$ cells. All coming definitions can then be easily extended to talk about space complexity.

As we want to talk about general functions in \mathcal{F} , we extend the approach to more general functions. (for conciseness, when $\mathbf{x} = (x_1, \dots, x_p)$, $\mathbf{X} = (X_1, \dots, X_p)$, we write $\mathbf{x} \in [-2^{\mathbf{X}}, 2^{\mathbf{X}}]$ as a shortcut for $x_1 \in [-2^{X_1}, 2^{X_1}], \dots, x_p \in [-2^{X_p}, 2^{X_p}]$).

Definition A.5 (Complexity for real functions: general case) Consider a function $f(x_1, \dots, x_p, n_1, \dots, n_q)$ whose domain is $\mathbb{R}^p \times \mathbb{N}^q$. We say that the (non-uniform) time complexity of f is bounded by a function $t' : \mathbb{N}^{p+q+1} \rightarrow \mathbb{N}$ if the time complexity of $f(\cdot, \dots, \cdot, \ell(n_1), \dots, \ell(n_q))$ on $[-2^{X_1}, 2^{X_1}] \times \dots \times [-2^{X_p}, 2^{X_p}]$

is bounded by a function $t(\cdot, \dots, \cdot, \ell(n_1), \dots, \ell(n_q), \cdot) : \mathbb{N}^p \times \mathbb{N} \rightarrow \mathbb{N}$ such that $t(\mathbf{x}, \ell(n_1), \dots, \ell(n_q), n) \leq t'(\mathbf{x}, \ell(n_1), \dots, \ell(n_q), n)$ whenever $\mathbf{x} \in [-2^{\mathbf{X}}, 2^{\mathbf{X}}]$. We say that f is polynomial time computable if t' can be chosen as a polynomial. We say that a vectorial function is polynomial time computable iff all its components are.

Remark 17 There is some important subtlety: when considering $f : \mathbb{N} \rightarrow \mathbb{Q}$, as $\mathbb{Q} \subset \mathbb{R}$, stating f is computable may mean two things: in the classical sense, given integer y , i.e. one can compute p_y and q_y some integers such that $f(y) = p_y/q_y$, or that it is computable in the sense of computable analysis: given some precision n , given arbitrary y , and n we can provide some rational (or even dyadic) q_n such that $|q_n - f(y)| \leq 2^{-n}$. As we said, we always consider the latter.

We do that so this measure of complexity extends the usual complexity measure for functions over the integers, where complexity of integers is measured with respects of their lengths, and over the reals, where complexity is measured with respect to their approximation. In particular, in the specific case of a function $f : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$, that basically means there is some polynomial $t' : \mathbb{N}^{d+1} \rightarrow \mathbb{N}$ so that the time complexity of producing some dyadic approximating $f(\mathbf{m})$ at precision 2^{-n} is bounded by $t'(\ell(m_1), \dots, \ell(m_d), n)$.

In other words, when considering that a function is polynomial time computable, it is in the length of all its integer arguments, as this is the usual convention.

B Some results from [8, 9, 3]

B.1 Some general statements from [8, 9]

To be self-contained, we recall in this section some results and concepts from [8, 9]. All the statements in this section are already present in [8, 9]: we are just repeating them here in case this helps. We provide some of the proofs when they are not in the preliminary ArXiv version.

As said in the introduction:

Definition B.1 (Discrete Derivative) The discrete derivative of $\mathbf{f}(x)$ is defined as $\Delta\mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x)$. We will also write \mathbf{f}' for $\Delta\mathbf{f}(x)$ to help readers not familiar with discrete differences to understand statements with respect to their classical continuous counterparts.

Several results from classical derivatives generalise to the settings of discrete differences: this includes linearity of derivation $(a \cdot f(x) + b \cdot g(x))' = a \cdot f'(x) + b \cdot g'(x)$, formulas for products and division such as $(f(x) \cdot g(x))' = f'(x) \cdot g(x+1) + f(x) \cdot g'(x) = f(x+1)g'(x) + f'(x)g(x)$. Notice that, however, there is no simple equivalent of the chain rule, in other words, there is no simple formula for the derivative of the composition of two functions.

A fundamental concept is the following:

Definition B.2 (Discrete Integral) Given some function $\mathbf{f}(x)$, we write

$$\int_a^b \mathbf{f}(x) \delta x$$

as a synonym for $\int_a^b \mathbf{f}(x) \delta x = \sum_{x=a}^{x=b-1} \mathbf{f}(x)$ with the convention that it takes value 0 when $a = b$ and $\int_a^b \mathbf{f}(x) \delta x = -\int_b^a \mathbf{f}(x) \delta x$ when $a > b$.

The telescope formula yields the so-called Fundamental Theorem of Finite Calculus:

Theorem B.3 (Fundamental Theorem of Finite Calculus) Let $\mathbf{F}(x)$ be some function. Then,

$$\int_a^b \mathbf{F}'(x) \delta x = \mathbf{F}(b) - \mathbf{F}(a).$$

A classical concept in discrete calculus is the one of falling power defined as

$$x^m = x \cdot (x-1) \cdot (x-2) \cdots (x-(m-1)).$$

This notion is motivated by the fact that it satisfies a derivative formula $(x^m)' = m \cdot x^{m-1}$ similar to the classical one for powers in the continuous setting. In a similar spirit, we introduce the concept of falling exponential.

Definition B.4 (Falling exponential) Given some function $\mathbf{U}(x)$, the expression \mathbf{U} to the falling exponential x , denoted by $\bar{\mathbf{2}}^{\mathbf{U}(x)}$, stands for

$$\begin{aligned} \bar{\mathbf{2}}^{\mathbf{U}(x)} &= (1 + \mathbf{U}'(x-1)) \cdots (1 + \mathbf{U}'(1)) \cdot (1 + \mathbf{U}'(0)) \\ &= \prod_{t=0}^{t=x-1} (1 + \mathbf{U}'(t)), \end{aligned}$$

with the convention that $\prod_0^0 = \prod_0^{-1} = \mathbf{id}$, where \mathbf{id} is the identity (sometimes denoted 1 hereafter).

This is motivated by the remarks that $2^x = \bar{2}^x$, and that the discrete derivative of a falling exponential is given by

$$(\bar{2}^{\mathbf{U}(x)})' = \mathbf{U}'(x) \cdot \bar{2}^{\mathbf{U}(x)}$$

for all $x \in \mathbb{N}$.

Lemma B.5 (Derivation of an integral with parameters) Consider

$$\mathbf{F}(x) = \int_{a(x)}^{b(x)} \mathbf{f}(x, t) \delta t.$$

Then

$$\mathbf{F}'(x) = \int_{a(x)}^{b(x)} \frac{\partial \mathbf{f}}{\partial x}(x, t) \delta t + \int_0^{-a'(x)} \mathbf{f}(x+1, a(x+1)+t) \delta t + \int_0^{b'(x)} \mathbf{f}(x+1, b(x)+t) \delta t.$$

In particular, when $a(x) = a$ and $b(x) = b$ are constant functions, $\mathbf{F}'(x) = \int_a^b \frac{\partial \mathbf{f}}{\partial x}(x, t) \delta t$, and when $a(x) = a$ and $b(x) = x$, $\mathbf{F}'(x) = \int_a^x \frac{\partial \mathbf{f}}{\partial x}(x, t) \delta t + \mathbf{f}(x+1, x)$.

Proof

$$\begin{aligned} \mathbf{F}'(x) &= \mathbf{F}(x+1) - \mathbf{F}(x) \\ &= \sum_{t=a(x+1)}^{b(x+1)-1} \mathbf{f}(x+1, t) - \sum_{t=a(x)}^{b(x)-1} \mathbf{f}(x, t) \\ &= \sum_{t=a(x)}^{b(x)-1} (\mathbf{f}(x+1, t) - \mathbf{f}(x, t)) + \sum_{t=a(x+1)}^{t=a(x)-1} \mathbf{f}(x+1, t) + \sum_{t=b(x)}^{b(x+1)-1} \mathbf{f}(x+1, t) \\ &= \sum_{t=a(x)}^{b(x)-1} \frac{\partial \mathbf{f}}{\partial x}(x, t) + \sum_{t=a(x+1)}^{t=a(x)-1} \mathbf{f}(x+1, t) + \sum_{t=b(x)}^{b(x+1)-1} \mathbf{f}(x+1, t) \\ &= \sum_{t=a(x)}^{b(x)-1} \frac{\partial \mathbf{f}}{\partial x}(x, t) + \sum_{t=0}^{t=-a(x+1)+a(x)-1} \mathbf{f}(x+1, a(x+1)+t) \\ &\quad + \sum_{t=0}^{b(x+1)-b(x)-1} \mathbf{f}(x+1, b(x)+t). \end{aligned}$$

□

Lemma B.6 (Solution of linear ODE) For matrices \mathbf{A} and vectors \mathbf{B} and \mathbf{G} , the solution of equation $\mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$

with initial conditions $\mathbf{f}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y})$ is

$$\begin{aligned}\mathbf{f}(x, \mathbf{y}) &= \left(\bar{2}^{\int_0^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{G}(\mathbf{y}) \\ &\quad + \int_0^x \left(\bar{2}^{\int_{u+1}^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \delta u.\end{aligned}$$

Proof Denoting the right-hand side by $\mathbf{rhs}(x, \mathbf{y})$, we have

$$\begin{aligned}\overline{\mathbf{rhs}}'(x, \mathbf{y}) &= \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \left(\bar{2}^{\int_0^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{G}(\mathbf{y}) \\ &\quad + \int_0^x \left(\bar{2}^{\int_{u+1}^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right)' \cdot \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \delta u \\ &\quad + \left(\bar{2}^{\int_{x+1}^{x+1} \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \\ &= \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \left(\bar{2}^{\int_0^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{G}(\mathbf{y}) \\ &\quad + \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \\ &\quad \quad \int_0^x \left(\bar{2}^{\int_{u+1}^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \delta u \\ &\quad + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \\ &= \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{rhs}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})\end{aligned}$$

where we have used linearity of derivation and definition of falling exponential for the first term, and derivation of an integral (Lemma B.5) providing the other terms to get the first equality, and then the definition of falling exponential. This proves the property by unicity of solutions of a discrete ODE, observing that $\overline{\mathbf{rhs}}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y})$. \square

We write also 1 for the identity.

Remark 18 Notice that this can be rewritten as

$$(B-1) \quad \mathbf{f}(x, \mathbf{y}) = \sum_{u=-1}^{x-1} \left(\prod_{t=u+1}^{x-1} (1 + \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y})) \right) \cdot \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}),$$

with the (not so usual) conventions that for any function $\kappa(\cdot)$, $\prod_x^{x-1} \kappa(x) = 1$ and $\mathbf{B}(-1, \mathbf{y}) = \mathbf{G}(\mathbf{y})$. Such equivalent expressions both have a clear computational content. They can be interpreted as an algorithm unrolling the computation of $\mathbf{f}(x+1, \mathbf{y})$ from the computation of $\mathbf{f}(x, \mathbf{y}), \mathbf{f}(x-1, \mathbf{y}), \dots, \mathbf{f}(0, \mathbf{y})$.

A fundamental fact is that the derivation with respects to length provides a way to do a kind of change of variables. This is Lemma 2.5 in the body of this article.

C Some statements from [3]

We repeat here some statements from [3]. They are motivated by repeating the arguments for proving Proposition 6.1, 1.

C.1 On the complexity of solving a linear length ODE

We have to prove that all functions of \mathbb{LDL}^\bullet are computable (in the sense of computable analysis) in polynomial time. The hardest part is to prove that the class of polynomial time computable functions is preserved by the linear length ODE schema, so we start by it.

Lemma C.1 *The class of polynomial time computable functions is preserved by the linear length ODE schema.*

We write $\|\cdot\cdot\cdot\|$ for the sup norm of integer part: given some matrix $\mathbf{A} = (A_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$, $\|\mathbf{A}\| = \max_{i,j} \lceil A_{i,j} \rceil$. In particular, given a vector \mathbf{x} , it can be seen as a matrix with $m = 1$, and $\|\mathbf{x}\|$ is the sup norm of the integer part of its components.

Before proving this lemma, we prove the following result:

Lemma C.2 (Fundamental observation) *Consider the ODE*

$$(C-1) \quad \mathbf{F}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{F}(x, \mathbf{y}) + \mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}).$$

Assume:

- The initial condition $\mathbf{G}(\mathbf{y}) =^{\text{def}} \mathbf{F}(0, \mathbf{y})$ is polynomial time computable, and $\mathbf{h}(x, \mathbf{y})$ are polynomial time computable with respect to the value of x .
- $\mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ are $\overline{\text{cond}}$ -polynomial expressions essentially constant in $\mathbf{F}(x, \mathbf{y})$.

Then, there exists a polynomial p such that $\ell(\|\mathbf{F}(x, \mathbf{y})\|) \leq p(x, \ell(\|\mathbf{y}\|))$ and $\mathbf{F}(x, \mathbf{y})$ is polynomial time computable with respect to the value of x .

Proof The solution of ordinary differential equation (C-1) can be put in some explicit form (this follows from [8, 9], or alternatively Remark 18 following Lemma B.6 in appendix, or, if you prefer, can be directly established by induction, using the recurrence formula (C-1)).

$$(C-2) \quad \mathbf{F}(x, \mathbf{y}) = \sum_{u=-1}^{x-1} \left(\prod_{t=u+1}^{x-1} (1 + \mathbf{A}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y})) \right) \cdot \mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}).$$

with the conventions that $\prod_x^{x-1} \kappa(x) = 1$ and $\mathbf{B}(\cdot, -1, \mathbf{y}) = \mathbf{G}(\mathbf{y})$.

This formula permits to evaluate $\mathbf{F}(x, \mathbf{y})$, using a dynamic programming approach, from the quantities \mathbf{y} , u , $\mathbf{h}(u, \mathbf{y})$, for $1 \leq u < x$, in a number of arithmetic steps that is polynomial in x . Indeed: for any $-1 \leq u \leq x$, $\mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ are matrices whose coefficients are $\overline{\text{cond}}\text{-polynomial}$. Their coefficients involve finitely many arithmetic operations or $\overline{\text{cond}}()$ operations from their inputs. Once this is done, computing $\mathbf{F}(x, \mathbf{y})$ requires polynomially in x many arithmetic operations: basically, once the values for \mathbf{A} and \mathbf{B} are known we have to sum up $x + 1$ terms, each of them involving at most $x - 1$ multiplications.

We need to take care not only on the arithmetic complexity that is polynomial in x , but also on the bit complexity. We start by discussing the bit complexity of the integer parts. Each of the quantities \mathbf{y} , u , $\mathbf{h}(u, \mathbf{y})$, for $1 \leq u < x$, being polynomial time computable, has its integer part with a bit complexity that remains polynomial in x and $\ell(\|\mathbf{y}\|)$. As the bit complexity of a sum, product, etc is polynomial in the size of its arguments, it is sufficient to show that the growth rate of function $\mathbf{F}(x, \mathbf{y})$ can be polynomially dominated. For this, recall that, for any $-1 \leq u \leq x$, coefficients of $\mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ are essentially constant in $\mathbf{F}(u, \mathbf{y})$. Hence, the size of the integer part of these coefficients do not depend on $\ell(\mathbf{F}(u, \mathbf{y}))$. Since, in addition, \mathbf{h} is computable in polynomial time in x and $\ell(\mathbf{y})$, there exists a polynomial p_M such that:

(C-3)

$$\max(\ell(\|\mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})\|), \ell(\|\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})\|) \leq p_M(u, \ell(\|\mathbf{y}\|))).$$

It then holds that,

$$\ell(\|\mathbf{F}(x + 1, \mathbf{y})\|) \leq p_M(x, \ell(\|\mathbf{y}\|)) + \ell(\|\mathbf{F}(x, \mathbf{y})\|) + 2$$

It follows from an easy induction that we must have

$$\ell(\|\mathbf{F}(x, \mathbf{y})\|) \leq \ell(\|G(\mathbf{y})\|) + x \cdot (p_M(x, \ell(\|\mathbf{y}\|)) + 2),$$

which gives the desired bound on the length of the integer part of the values for function \mathbf{F} , after observing that, since \mathbf{G} is polynomial time computable, necessarily $\ell(\|G(\mathbf{y})\|)$ remains polynomial in $\ell(\|\mathbf{y}\|)$.

We now take care of the bit complexity of involved quantities, in order to prove that $\mathbf{F}(x, \mathbf{y})$ is indeed polynomial time computable with respect to the value of x . Given \mathbf{y} , we can determine some integer Y such that $\mathbf{y} \in [2^{-Y}, 2^Y]$. We just need to prove that given n , we can provide some dyadic \mathbf{z}_x approximating $\mathbf{F}(x, \mathbf{y})$ at precision n , i.e. with $\|\mathbf{F}(x, \mathbf{y}) - \mathbf{z}_x\| \leq 2^{-n}$, in a time polynomial in x and Y .

Basically, this follows from the possibility of evaluating $\mathbf{F}(x, \mathbf{y})$ using formula (C-2). This latter formula is made of a sum of $x+1$ terms, that we can call $\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_x$, each of them \mathbf{T}_i corresponding to a product of k matrices (or vectors) $\mathbf{T}_i = \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)}$, with $k(i) \leq x+1$, were each \mathbf{C}_j is either some $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ for some u or some $(1 + \mathbf{A}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}))$ for some t .

To solve our problem, it is sufficient to be able to approximate the value of each \mathbf{T}_i by some dyadic \mathbf{d}_i with precision 2^{-n-m} , considering m with $x+1 \leq 2^m$. Indeed, taking $\mathbf{z}_x = \sum_{i=0}^x \mathbf{d}_i$ will guarantee an error on the approximation of $\mathbf{F}(x, \mathbf{y})$ less than $(x+1)2^{-n-m} \leq 2^{-n}$.

So we focus on the problem of estimating $\mathbf{T}_i = \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)}$ with precision 2^{-n-m} . If we write $\tilde{\mathbf{C}}_j$ for some approximation of \mathbf{C}_j , we can write

$$\begin{aligned}
 (C-4) \quad & \|\mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_1 \tilde{\mathbf{C}}_2 \dots \tilde{\mathbf{C}}_{k(i)}\| \leq \\
 & + \|\mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-1} (\mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_{k(i)})\| \\
 & + \|\mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-2} (\mathbf{C}_{k(i)-1} - \tilde{\mathbf{C}}_{k(i)-1}) \tilde{\mathbf{C}}_{k(i)}\| \\
 & + \|\mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-3} (\mathbf{C}_{k(i)-2} - \tilde{\mathbf{C}}_{k(i)-2}) \tilde{\mathbf{C}}_{k(i)-1} \tilde{\mathbf{C}}_{k(i)}\| \\
 & \vdots \\
 & + \|(\mathbf{C}_1 - \tilde{\mathbf{C}}_1) \tilde{\mathbf{C}}_1 \tilde{\mathbf{C}}_2 \dots \tilde{\mathbf{C}}_{k(i)}\|.
 \end{aligned}$$

We just need then to compute some approximation $\tilde{\mathbf{C}}_{k(i)}$ of $\mathbf{C}_{k(i)}$, guaranteeing the first term in (C-4) to be less than 2^{-n-m-m} , and then choose some approximation $\tilde{\mathbf{C}}_{k(i)-1}$ of $\mathbf{C}_{k(i)-1}$, guaranteeing the second term in (C-4) to be less than 2^{-n-m-m} , and so on. Doing so, we will solve our problem, as $\tilde{\mathbf{C}}_1 \tilde{\mathbf{C}}_2 \dots \tilde{\mathbf{C}}_{k(i)}$ will provide an estimation of \mathbf{T}_i , with an error less than $(x+1)2^{-n-m-m} \leq 2^{-n-m}$ by (C-4).

For the first term of (C-4), the point is that we know from a reasoning similar to previous computations (namely bounds such as (C-3)) that we have

$$\|\mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-1}\| \leq 2^{p_1(x, Y)}$$

for some polynomial p_1 .

Consequently, it is sufficient to take $\|\mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_{k(i)}\| \leq 2^{-n-2m-p_1(x, Y)}$ to guarantee an error less than 2^{-n-m-m} .

A similar analysis applies for the second, and other terms, that involve finitely many terms. The whole approach takes a time that remains clearly polynomial in x and Y . \square

The previous statements lead to the following:

Lemma C.3 (Intrinsic complexity of linear \mathcal{L} -ODE) *Let \mathbf{f} be a solution of the linear \mathcal{L} -ODE*

$$\begin{aligned}\mathbf{f}(0, \mathbf{y}) &= \mathbf{g}(\mathbf{y}), \\ \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} &= \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})\end{aligned}$$

where \mathbf{u} is essentially linear in $\mathbf{f}(x, \mathbf{y})$. Assume that the functions $\mathbf{u}, \mathbf{g}, \mathbf{h}$ are computable in polynomial time. Then, \mathbf{f} is computable in polynomial time.

Proof From Lemma 2.5, $\mathbf{f}(x, \mathbf{y})$ can also be given by $\mathbf{f}(x, \mathbf{y}) = \bar{\mathbf{F}}(\ell(x), \mathbf{y})$ where $\bar{\mathbf{F}}$ is the solution of the initial value problem

$$\begin{aligned}\bar{\mathbf{F}}(1, \mathbf{y}) &= \mathbf{g}(\mathbf{y}), \\ \frac{\partial \bar{\mathbf{F}}(t, \mathbf{y})}{\partial t} &= \mathbf{u}(\bar{\mathbf{F}}(t, \mathbf{y}), 2^t - 1, \mathbf{y}).\end{aligned}$$

Functions \mathbf{u} are $\overline{\text{cond}}$ -polynomial expressions that are essentially linear in $\mathbf{f}(x, \mathbf{y})$. So there exist matrices \mathbf{A}, \mathbf{B} that are essentially constants in $\mathbf{f}(t, \mathbf{y})$ such that

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}).$$

In other words, it holds

$$\mathbf{F}'(t, \mathbf{y}) = \bar{\mathbf{A}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) \cdot \mathbf{F}(t, \mathbf{y}) + \bar{\mathbf{B}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}).$$

by setting

$$\begin{aligned}\bar{\mathbf{A}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) &= \mathbf{A}(\bar{\mathbf{F}}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y}) \\ \bar{\mathbf{B}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) &= \mathbf{B}(\bar{\mathbf{F}}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y})\end{aligned}$$

The corresponding matrix $\bar{\mathbf{A}}$ and vector $\bar{\mathbf{B}}$ are essentially constant in $\mathbf{F}(t, \mathbf{y})$. Also, functions \mathbf{g}, \mathbf{h} are computable in polynomial time, more precisely polynomial in $\ell(x)$, hence in t , and $\ell(y)$. Given t , obtaining $2^t - 1$ is immediate. This guarantees that all hypotheses of Lemma C.2 are true. We can then conclude remarking, again, that $t = \ell(x)$. \square

This proves Lemma C.1.

C.2 Proof of Proposition 6.1, 1.

We can now state and prove Proposition 6.1, 1:

Proof of Proposition 6.1, 1. This is proved by induction. This is true for basis functions, from basic arguments from computable analysis. In particular as $\overline{\text{cond}}$ is a continuous piecewise affine function with rational coefficients, it is computable in polynomial time from standard arguments.

Now, the class of polynomial time computable functions is preserved by composition. This is proved in [20]: in short, the idea of the proof for $\text{COMP}(f, g)$, is that by induction hypothesis, there exists M_f and M_g two Turing machines computing in polynomial time $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$. In order to compute $\text{COMP}(f, g)(x)$ with precision 2^{-n} , we just need to compute $g(x)$ with a precision $2^{-m(n)}$, where $m(n)$ is the polynomial modulus function of f . Then, we compute $f(g(x))$, which, by definition of M_f takes a polynomial time in n . Thus, since polynomial time with an oracle in polynomial time is polynomial time, $\text{COMP}(f, g)$ is computable in polynomial time, so the class of polynomial time computable functions is preserved under composition. It only remains to prove that the class of polynomial time computable functions is preserved by the linear length ODE schema: this is Lemma C.1. \square

Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

Manon.Blanc@lix.polytechnique.fr, [Olivier.Bournez@polytechnique.fr](mailto>Olivier.Bournez@polytechnique.fr)

<https://perso.crans.org/mblanc/>, <https://www.lix.polytechnique.fr/~bournez/>

Received: aa bb 20YY Revised: cc dd 20ZZ